Desarrollo de una herramienta Data Quality para entornos analíticos

Trabajo de Final de Grado

Ignacio N. Lucero Ascencio

Universidad Politécnica de Catalunya

Facultad de Informática de Barcelona

Especialidad en Ingeniera del Software

Director
Marc Ruiz Figueras

Tutor
Óscar Romero Moral

19 de enero de 2016

Resumen

El objetivo del presente proyecto es sencillo: a partir de una herramienta de calidad de datos existente, se trata de crear una versión nueva, mucho más eficiente y comercializable, utilizando tecnologías más actuales y apropiadas. La versión nueva deberá ser más usable, estética, fácil de mantener, portable, robusta y adaptable.

Con ese objetivo siempre como meta, en esta memoria se sintetiza la totalidad del proceso de gestión y desarrollo del proyecto. Empezando por la definición del problema y el estudio de mercado, justificamos la necesidad de este proyecto. Definiendo el alcance, planificación temporal y gestión económica, constatamos que es viable emprenderlo. Finalmente, en las etapas de especificación, arquitectura e implementación, desarrollamos la aplicación y razonamos sobre las decisiones de diseño tomadas.

Concluimos la disertación constatando el éxito del proyecto, ya que la nueva versión de la herramienta, aun contando con un subconjunto (nada despreciable) de las funcionalidades de la versión original, funciona perfectamente sobre las tecnologías elegidas y cumple plenamente con los objetivos de mejora propuestos.

Resum

L'objectiu d'aquest projecte és senzill: a partir d'una eina de qualitat de dades existent, volem crear una nova versió, molt més eficient i comercialitzable, tot fent servir tecnologies més actuals i apropiades. La nova versió haurà de ser més usable, estètica, fàcil de mantenir, portable, robusta i adaptable.

Amb aquest objectiu sempre en ment, en aquesta memòria es sintetitza la totalitat del procés de gestió y desenvolupament del projecte. Començant per la definició del problema i l'estudi de mercat, justifiquem la necessitat del projecte. Definint l'abast, la planificació temporal y la gestió econòmica, constatem que es viable emprendre'l. Finalmente, en les etapes d'especificació, arquitectura i implementació, desenvolupem la aplicació y raonem sobre les decisions de disseny preses.

Acabem la dissertació constatant l'èxit del projecte, ja que la nova versió de l'eina, encara que consta d'un subconjunt (no gens menyspreable) de les funcionalitats de la versió original, funciona perfectament sobre les tecnologies triades i compleix plenament amb els objectius de millora proposats.

Abstract

The main goal of this project is simple: taking an existing data quality tool as a starting point, we aim to create a new version, much more efficient and marketable, using more suitable and up-to-date technologies. The new version shall be more usable, aesthetic, maintainable, portable, robust and responsive.

Keeping this goal in mind, in this report we can find the whole process of management and development of the project. Starting by the definition of the problem and the market analysis, we justify the need for this project. By defining the scope, schedule and economical management, we establish the viability of tackling it. Finally, in the stages of specification, architecture and implementation, we develop the application and reason about the design decisions that we have made.

We conclude the dissertation by confirming the success of the project, given that the new version of the tool, even when implementing a (not insignificant) subset of the functionalities of the original version, works perfectly upon the chosen technologies and fully accomplishes the improvement goals that have been set.

Agradecimientos

Quiero agradecer a mi ponente Óscar Romero por sus valiosas indicaciones y sugerencias para documentar el proyecto, y a mi director Marc Ruiz por haberme guiado a lo largo del mismo.

Finalmente, quiero agradecer a mi amigo Pere por la ayuda que me ha brindado revisando los esbozos finales de la memoria.

Índice general

Índice de figuras

Índice de cuadros

1 Introducción

Aunque para cualquier empresa siempre ha sido necesario dar importancia a los datos e indicadores relacionados con su negocio, hoy más que nunca las empresas entienden que esta es una parte fundamental del éxito [20]. Los datos constituyen una sólida base desde la cual tomar decisiones estratégicas y adaptar los servicios a las volubles necesidades de los clientes. Por otro lado, una pobre calidad en los datos puede afectar negativamente la productividad y competitividad de una organización. Ya sea para una decisión simple como ajustar la compra de materia prima o para algo más complejo como estimar una proyección futura de la demanda, será necesario tener una muestra significativa de datos, que cumplan con unas normas de calidad definidas, y tener buenas herramientas y algoritmos para analizarlos. Cumpliendo estas tres condiciones se puede afrontar el desafío de realizar previsiones y tomar decisiones corporativas que estén encaminadas hacia el progreso y éxito de la empresa.

En un mundo en el que el coste de almacenamiento en disco es cada año más barato [11] y la cantidad de información generada cada día es inmensa, no resulta difícil conseguir una gran cantidad de datos para analizar. El desafío reside en que los datos sean significativos y de buena calidad, y en la habilidad para analizarlos y extraer información a partir de ellos. El presente trabajo abarca solamente aspectos relacionados con el primer desafío, la calidad de los datos, en un entorno concreto: la herramienta *iQuality*, que presentaremos en la Sección 1.2.

1.1 ¿Qué es la calidad de datos?

Se puede definir la calidad de datos como el grado de excelencia que presenta un dato en relación con con lo que representa [3]. Alternativamente, otras definiciones ponen más énfasis en la utilidad del dato para tomar decisiones que en su capacidad de representar una entidad física o relación: los datos son de calidad si "se adecúan a su uso esperado en cuanto a operaciones, toma de decisiones y planificación" [9]. Podríamos argumentar que si un dato no se ajusta a la realidad no puede ser útil, con lo cual las dos definiciones resultarían equivalentes.

Aunque existen diversos enfoques para determinar la calidad de los datos [12, 10], hay un consenso en cuanto al proceso general: definir las reglas[1] que hay que aplicar, validarlas sobre los datos, y resolver las incoherencias que puedan surgir. Es decisivo que las reglas se definan

[1] Usaremos de forma equivalente los términos *reglas* y *certificaciones* para hablar de las condiciones que los datos deben cumplir para ser de calidad.

en el contexto del negocio: cada proyecto tiene sus propios requerimientos y restricciones, y en general serán diferentes a los de otros.

Las reglas no son más que condiciones que los datos deben cumplir para ser de calidad. Las hay de muchos tipos, pero los esenciales son:

Precisión: hasta qué punto el dato es una representación fiel del concepto del mundo real que simboliza.

Exactitud: el nivel de exactitud con el que el dato se ajusta a la medida real que representa.

Accesibilidad: la facilidad con la que los usuarios pueden acceder al dato, sin desperdiciar tiempo ni recursos.

Consistencia: se cumple cuando los datos son idénticos para cualquier proceso o unidad de la organización.

Exhaustividad: todos los datos representativos de una entidad son capturados y representados.

Actualidad: los datos deben estar actualizados y ser representativos del período que se quiere analizar.

Temporalidad: los datos están disponibles en el momento que se necesitan.

Definición: cada elemento debe tener una definición precisa, de manera que los usuarios entiendan lo que representa un dato y lo manipulen adecuadamente.

Algunas de estas características pueden parecer evidentes y fáciles de satisfacer, pero cumplirlas no resulta trivial en grandes organizaciones que manipulan y almacenan ingentes cantidades de datos. Éstos pasan por varios procesos y unidades desde que son insertados en el sistema hasta que son analizados, y provienen de fuentes muy variadas y dispares. Durante ese camino, muchos factores pueden afectar a la calidad e integridad de los datos: desde un dato mal introducido por un simple error humano, hasta la falta de algún campo relevante de algunas entidades, pasando por datos duplicados y relaciones incorrectas.

Otros conceptos importantes relacionados con la calidad de datos son el **gobierno** de datos y la **trazabilidad** de los datos. El gobierno de datos tiene por objeto asegurarse de que estos son siempre fiables y válidos en cada contexto, que su calidad se mantiene a lo largo del tiempo y que existen mecanismos de control sobre quién puede hacer determinadas operaciones sobre los datos en cada momento. Esto permite hacer de los datos un activo corporativo de gran valor empresarial. La trazabilidad de los datos permite poder mostrar (y demostrar) cómo se ha calculado un determinado indicador, en base a qué datos, aplicando qué criterios, quién lo ha calculado y cuándo. En definitiva, poder seguir la evolución temporal de los datos e indicadores a lo largo de todas sus transformaciones. De hecho, la trazabilidad de los datos es obligatoria

para empresas como aseguradoras y entidades financieras, ya que tienen que rendir cuentas a instituciones reguladoras nacionales y europeas [4].

Dada la complejidad asociada a los conceptos descritos, es natural que existan y se valoren mucho las herramientas *software* para mantener y mejorar la calidad de los datos. Para ello se creó la herramienta *iQuality*.

1.2 Descripción de la solución iQuality

Para cubrir todas las necesidades de calidad de datos de la entidad VidaCaixa[2], el año 2012 Indra crea la herramienta *software iQuality*. Las principales necesidades que cubre para los empleados de VidaCaixa son:

- Planificar, realizar y gestionar la extracción, transformación y carga de datos (ETL).
- Generar informes periódicos de los indicadores relevantes para su negocio.
- Demostrar el origen de indicadores mediante un diccionario de conceptos.
- Poder devolver el sistema a un estado pasado para contrastar el cálculo de indicadores ante una institución reguladora.
- Monitorizar el estado de la calidad de los datos.
- Parametrizar las certificaciones y reglas que afectan a la determinación de la calidad de los datos.

Los usuarios pueden hacer todo esto sin tener conocimientos técnicos más allá de los del propio negocio.

Por otro lado, la herramienta también tiene utilidad para los empleados de Indra, ya que les permite detectar errores durante un proceso ETL, relanzar cargas fallidas y asesorar en cuanto a la calidad de datos del sistema, entre muchas otras funcionalidades.

En cuanto a la arquitectura actual[3] de *iQuality*, podemos discernir tres partes fundamentales:

1. Un Data mart con los datos de negocio y procedimientos almacenados escritos en PL/SQL que se encargan de hacer transformaciones sobre los datos.

[2] Para más información sobre VidaCaixa, véase [22].

[3] A lo largo de este documento, nos referiremos a la versión de *iQuality* que queremos mejorar mediante el presente proyecto como la *versión actual* (ya que es la que se sigue usando actualmente) o *versión anterior* (ya que es la versión previa a la que estamos implementando). En cambio, nos referiremos a la versión cuyo desarrollo es el objeto de este proyecto como la *versión nueva*.

2. Un servidor que lanza procesos Unix sobre el Data mart y al cual el administrador se puede conectar para manipular los datos.

3. Una aplicación web que permite a los usuarios utilizar las funcionalidades de *iQuality*.

La tercera parte es la que concierne a este proyecto, como se verá en más detalle en el Capítulo 3. Dentro de la misma, podemos distinguir cuatro bloques funcionales diferentes[4]:

1. **Control de procesos de carga.** Engloba las tareas asociadas a un proceso ETL. En el contexto de *iQuality*, se llama *pase* a la definición en el sistema de un proceso ETL y *ejecución* a una instancia de un pase en un mes y año concretos. Un pase consta de varios *jobs*, que son partes atómicas de un proceso ETL (e.g., carga de un fichero o comprobación de que alguna tabla existe). Entre jobs de un pase se pueden definir *dependencias*, de forma que un job no puede empezar hasta que todos los jobs de los que depende no se hayan ejecutado con éxito. Este bloque cubre las siguientes funcionalidades:

   a) Consulta de las certificaciones disponibles. Perfil necesario: validación.

   b) Consulta del resultado de las certificaciones a nivel global y a nivel detallado individual. Perfil necesario: validación.

   c) Carga de tablas a partir de ficheros Excel del usuario. Perfil necesario: validación.

   d) Consulta de las ejecuciones planificadas para ser lanzadas y sus jobs asociados. Perfil necesario: validación.

   e) Consulta del estado de las ejecuciones lanzadas y cada uno de los jobs que las componen. Perfil necesario: validación.

   f) Alta, baja o modificación de una certificacioón. Perfil necesario: funcional.

   g) Parametrización de certificaciones. Perfil necesario: administrador.

   h) Alta, baja o modificación de pases, jobs y dependencias. Perfil necesario: administrador.

   i) Actualizar el estado de los pases o jobs, así como planificar nuevos pases a partir de otros ya definidos. Perfil necesario: administrador.

2. **Diccionario de conceptos.** El diccionario de conceptos se encarga de almacenar las definiciones asociadas a los elementos de información que se publican en el Data mart,

[4]Cada funcionalidad requiere de un nivel de privilegios mínimo. Los perfiles asociados a estos privilegios se describen en la Sección 1.3.

atendiendo a una estructuración de los mismos en base a indicadores y dimensiones (modelo dimensional). Cubre las siguientes funcionalidades:

a) Consulta de árbol de entidades, dimensiones e indicadores. Perfil necesario: validación.

b) Consulta de las fichas de definición de cada elemento terminal del árbol (atributos e indicadores). Perfil necesario: validación.

c) Modificación de los elementos del árbol. Perfil necesario: funcional.

d) Consulta de la traza asociada a cada elemento. Perfil necesario: validación.

e) Modificación de la traza asociada a cada elemento. Perfil necesario: administrador.

f) Consulta de las reglas de certificación asociadas a cada indicador. Perfil necesario: validación.

g) Alta, baja o modificación de las reglas de certificación asociadas a cada indicador. Perfil necesario: funcional.

h) Alta o baja de los elementos del árbol. Perfil necesario: administrador.

3. **Gestión de escenarios.** Los escenarios (o versiones de datos) son necesarios para crear simulaciones de distintos conjuntos de datos y poder comparar sus resultados. Este bloque consta de:

   a) Consulta de escenarios. Perfil necesario: funcional.

   b) Alta, baja y modificación de escenarios de datos del Data mart. Perfil necesario: administrador.

4. **Gestión de usuarios.** Acciones relacionadas con los usuarios y perfiles del sistema. En concreto:

   a) Consulta de usuarios y perfiles que acceden a *iQuality*. Perfil necesario: funcional.

   b) Alta, baja y modificación de usuarios y perfiles que acceden a *iQuality*. Perfil necesario: administrador.

1.3 Stakeholders

Es natural que un proyecto de esta envergadura involucre y afecte a varias entidades y personas. Dado que es muy importante tener en cuenta sus objetivos y expectativas, a continuación presentamos una breve descripción de cada una de las partes interesadas del proyecto.

1.3.1 VidaCaixa

Entidad financiera filial de *CaixaBank*. Para ella se diseñó e implementó el *software iQuality*, es decir, es el cliente.

1.3.2 Indra

Empresa multinacional consultora tecnológica. Es la empresa desarrolladora de *iQuality*.

1.3.3 Usuarios

iQuality admite tres niveles o perfiles de usuarios. Están organizados por jerarquías, de manera que un usuario de una jerarquía superior a otro puede realizar las mismas funciones que este más otras adicionales.

Usuario de validación. Típicamente se trata de un usuario de VidaCaixa. Puede consultar el diccionario de conceptos, consultar el estado de los datos y de las cargas del Data mart, así como cargar ficheros Excel.

Usuario funcional. Interlocutores definidos entre VidaCaixa e Indra. Además de todas la funcionalidades de un usuario de validación, puede modificar el diccionario de conceptos[5] y realizar consultas de otro módulos.

Usuario administrador. Es el encargado de la administración y mantenimiento de la aplicación. Puede consultar y modificar cualquiera de sus módulos, así como gestionar los privilegios de cualquier otro usuario.

[5] Realmente sólo puede realizar esta funcionalidad en el entorno de integración, en contraposición al administrador, que puede realizarla también en producción. Para más información sobre los entornos, véase la introducción del Capítulo 9.

1.3.4 Autor

Parte doblemente interesada, tanto como estudiante realizando el presente proyecto a modo de trabajo de final de grado, como desarrollador remunerado.

2 Estado del arte

Al ser la calidad de datos un aspecto tan importante para las grandes organizaciones, resulta muy natural que podamos encontrar en el mercado varias herramientas *data quality* corporativas: Oracle Enterprise Data Quality, IBM InfoSphere Information Server for Data Quality, SAP Data Quality Management o Informatica Data Quality, por nombrar algunas. Incluso hay soluciones *open source*, como DataCleaner. Es importante tener en cuenta las características de estas herramientas alternativas para evaluar la viabilidad del proyecto.

2.1 Estudio de mercado

Detengámonos por un momento a analizar las características de algunos de estos productos, de forma que tengamos una visión del estado actual de las herramientas *data quality*.

2.1.1 Oracle Enterprise Data Quality

Es una familia de productos que cubre varias de las necesidades de cualquier herramienta *data quality*: perfilado y auditoria de datos, para descubrir y cuantificar problemas y medir la calidad de los datos en función de las reglas de negocio; *parseado* y estandarización, para extraer información estructurada a partir de textos y tener todos los datos de un mismo tipo (teléfonos, direcciones, etc.) en un formato homogéneo; fusión y asociación de datos, para eliminar duplicados o evitarlos; consolidación e integración de datos de clientes a partir de plantillas de reglas; comprobación de direcciones; y otras funcionalidades más avanzadas como reconocimiento semántico de datos, clasificación y extracción de atributos, estandarización de descripciones, etc. [13]

2.1.2 IBM InfoSphere Information Server for Data Quality

Ofrece funcionalidades personalizables para la limpieza de datos, estandarizando información y automatizando la investigación y análisis de los datos en función de las reglas de negocio; monitoriza y mantiene la calidad de los datos, descubriendo problemas y proponiendo un plan de acción que se alinee con las reglas de negocio, al mismo tiempo que ofrece soporte en cuanto a gobierno de datos. Proporciona un entorno unificado que permite analizar la calidad de datos

de una manera ágil y enfocada al negocio, y permite definir reglas de validación y monitorizar su progreso [6].

2.1.3 SAP Data Quality Management

Hace hincapié en que los usuario puedan monitorizar la calidad y gobierno de datos mediante una interfaz agradable, además de: mejorar la calidad de los datos mediante *parseado* y estandarización de datos, mejorar la calidad de los datos existentes mediante información geoespacial, eliminar duplicados emparejando y consolidando datos a diferentes niveles de la lógica de negocio, y perfilar datos a partir de reglas de negocio [18].

2.1.4 SAS Data Quality

Presenta una característica interesante que no se encuentra en las otras herramientas: no nos limita a bases de datos relacionales, sino que también se puede integrar con bases de dato NoSQL tales como Hadoop o Impala. Esto representa otro grado de libertad para la herramienta, que se puede adaptar a entornos más diversos. Por supuesto, también ofrece las funcionalidades básicas de una herramienta de esta índole: limpieza de datos, eliminando duplicados y estandarizando; perfilado de datos, permitiendo establecer correlaciones ocultas entre entidades; monitorización y gobierno de datos a través de una interfaz web; integración de datos (ETL); obtención de una visión global de los datos a partir de diversas fuentes de un mismo negocio, y resolución de entidades a partir de diferentes entradas incompletas [19].

2.1.5 DemandTools

Esta herramienta es un poco más modesta que las de los grandes productores de *software* como Oracle o SAP, por lo tanto dispone de menos funcionalidades de monitorización y gobierno de datos. No obstante, sí que ofrece herramientas de mantenimiento, para realizar la carga y manipulación masiva de datos; funcionalidades de limpieza, para eliminar duplicados y fusionar datos; y utilidades de descubrimiento, para comparar datos externos con los almacenados antes de importar [1].

2.1.6 RingLead

Destaca por ser un servicio 100 % *cloud-based*. Esto puede ser bueno para un nuevo negocio, pero para una gran corporación la migración de los datos a la nube sería probablemente un proceso costoso y molesto. De todas formas, ofrece varias de las funcionalidades deseadas: limpieza y estandarización de datos, importación de datos sin duplicados, etc. Al igual que DemandTools, no parece centrarse especialmente en el gobierno de datos, validaciones ni trazabilidad [16].

2.1.7 Informatica Data Quality

Permite obtener una visión global de los datos y entender la relación que hay entre ellos; validar, desduplicar, estandarizar y consolidar datos; facilita el uso para usuarios del negocio mediante una interfaz adaptable, y les permite definir y probar reglas de negocio sin necesidad de los especialistas informáticos; facilita la construcción y mantenimiento de un glosario del negocio mediante la colaboración de todos los departamentos; y permite construir las reglas de negocio una vez y desplegarlas en diversas plataformas, como Hadoop o en la nube [7].

2.1.8 DataCleaner

Herramienta que destaca por ser *open source* y por ofrecer integración con diversas bases de datos NoSQL como MongoDB, elasticsearch y Cassandra. También dispone de las funcionalidades esenciales de una herramienta de este tipo: análisis y perfilado de datos, para descubrir patrones, datos que faltan y cualquier información que se pueda deducir a partir de los datos; detección y eliminación de duplicados; estandarización y limpieza de datos; y monitorización de la calidad a partir de las reglas de negocio y de validación definidas [2].

2.2 Conclusión del estudio de mercado

Las herramientas *data quality* hasta aquí analizadas son de carácter genérico: están pensadas para adaptarse a cualquier entorno corporativo. En cambio, la herramienta *iQuality* fue creada para satisfacer unas necesidades de calidad de datos concretas.

Por lo tanto, resulta natural mejorar y actualizar la versión actual de *iQuality* en vez de intentar adaptar una herramienta genérica. Al fin y al cabo, sólo se trata de mejorar ciertos aspectos de la herramienta a la vez que introducimos algunas nuevas funcionalidades. Adaptar una solución *data quality* genérica como las arriba estudiadas supondría un gran coste no sólo económico, si no de recursos y tiempo para hacer la migración y aprender a usar el nuevo *software*.

Por otro lado, Indra tiene intereses corporativos en producir una herramienta de calidad de datos propia ya que, aunque en esta instancia se aplica a una entidad en concreto, en un futuro interesa adaptarla a las necesidades de otros clientes. Todo esto justifica la iniciativa de emprender el proyecto de mejora de *iQuality*, y no la adaptación de otra herramienta comercial.

3 Alcance

3.1 Objetivos

Como se ha visto en la Sección 1.2, la herramienta *iQuality* consta de tres partes. Este proyecto sólo abarca la tercera de ellas, i.e., la aplicación web de *iQuality*.

Dicha aplicación ha sido implementada, tanto el *backend* como el *frontend*, mediante un *framework* propietario de Oracle llamado APplication EXpress (APEX). Aunque tiene una curva de aprendizaje baja y permite crear una aplicación de forma muy rápida, no brilla por su fluidez y escalabilidad. En su interior, todo está implementado mediante tablas relacionales. Esto no ofrece mucha flexibilidad a los desarrolladores, entre otras desventajas que veremos a continuación.

Con esto en mente, el **propósito final** del proyecto es migrar el *backend* de la aplicación web de APEX a Java, con el soporte del *framework* Spring, y el *frontend* a una implementación más moderna con librerías de JavaScript y el *framework* Bootstrap.

Como descubriremos en el Capítulo 4, probablemente no dispongamos del tiempo suficiente para migrar la aplicación entera dentro del período establecido. Por lo tanto, el proyecto se centrará en los dos primeros bloques funcionales descritos en la Sección 1.2: control de procesos de carga y diccionario de conceptos. Daremos prioridad a las funcionalidades de consulta de cada bloque, ya que resultan más sencillas, tras lo cual implementaremos las de alta, baja y modificación. Tampoco tendremos en cuenta en el presente proyecto el nivel de privilegios necesarios para utilizar cada funcionalidad.

Son varios los aspectos que esperamos mejorar con la migración de la aplicación[1]:

Usabilidad La aplicación actual no resulta fácil de usar: menús mal distribuidos, pantallas con mucha información, navegabilidad poco intuitiva, funcionalidades que requieren muchas interacciones, botones pequeños y difíciles de ver, etc. Podemos mejorar este diseño con Bootstrap como fundamento para construir una interfaz que mejore la experiencia del usuario.

[1] Varios de los términos siguientes no disponen de una traducción exacta al castellano o bien los vocablos recomendados no tienen la misma expresividad que en la voz inglesa, por lo que he mantenido la versión original para algunos de ellos.

look & feel El aspecto estético actual es modesto y poco atractivo: colores monótonos, relación entre las fuentes poco cuidadas, falta de elementos dinámicos con los que interactuar, etc. No es difícil mejorar este aspecto gracias a Bootstrap y varios *plug-in* basados en jQuery. Estas tecnologías ayudarán también a que la aplicación funcione correctamente en varios navegadores y sea responsive (se adapte a la resolución del dispositivo donde se está visualizando).

Responsivity (tiempo de respuesta) Este es uno de los aspectos claves que hay que mejorar. La implementación actual demora mucho en cargar algunas pantallas[2], especialmente la del diccionario de conceptos. Esto se puede mejorar enormemente cargando diferentes partes de una página asíncronamente gracias a tecnologías como AJAX, y cacheando astutamente algunos datos que son costosos de generar.

Mantenibilidad La implemetación en APEX tiene el gran inconveniente de mezclar el código que genera la vista con la lógica de la aplicación y las *queries* a la base de datos. Este acoplamiento resulta muy perjudicial para *debuggar* la aplicación o cuando deseamos cambiar alguna funcionalidad. Mejoraremos este aspecto aplicando el patrón Modelo Vista Controlador (MVC), mediante el cual podemos desacoplar las capas de vista, lógica y datos. Además, generaremos el Javadoc de la implementación Java (véase el Apéndice B), lo cual proporcionará una importante fuente de documentación a los siguientes desarrolladores que tengan que continuar en este proyecto.

Portabilidad Otro inconveniente de APEX es que, al ser un *software* propietario de Oracle, sólo funciona con una base de datos Oracle. Utilizando el patrón fachada desacoplaremos la capa de datos de la lógica. Esto proporcionará a *iQuality* una mayor abstracción, lo cual permitirá fácilmente utilizar cualquier sistema de base de datos (incluso NoSQL), siempre que disponga de un *driver* para Java.

Robustez La aplicación actual no ofrece ningún mecanismo de detección y tratamiento de errores, lo cual la hace poco robusta. Usaremos el tratamiento de excepciones propio de Java para atrapar y tratar posibles errores dinámicamente (especialmente todos los relacionados con la base de datos), y el sistema de *logging* logback para guardar un diario de las operaciones que realiza la aplicación. Además, utilizaremos el *framework* jUnit para añadir pruebas automatizadas a las clases implementadas. Finalmente, aprovecharemos el proceso de migración para detectar y corregir posibles inconsistencias en el modelo[3]. Todo esto hará de *iQuality* una herramienta más sólida, robusta, y dará más información a los administradores en caso de fallo.

Adicionalmente, sería interesante dotar a la herramienta de algunas funcionalidades nuevas que lleven la versión actual un paso más adelante. Concretamente, sería muy útil y atractivo disponer de un panel de control desde el cual monitorizar la calidad de datos del sistema y parametrizar

[2] ¡Hasta el punto de que algunas de ellas no cargan! El servidor tarda tanto en generar algunas pantallas que se activa un *timeout* antes de que el navegador pueda cargar la página.

[3] Entre los cuales ya se han identificado, por ejemplo, ciclos en las dependencias entre jobs, y creación de pases sin jobs asociados

las reglas que se aplican para calcular dicha calidad. Las funcionalidades se enumeran y detallan en el *backlog* que veremos en la Subsección 4.2.1.

Aspectos que quedan fuera del alcance del proyecto, pero que sería importante tener en cuenta para futuras versiones de *iQuality*, son:

Disponibilidad y durabilidad No está claro cómo respondería el sistema ante un fallo en la base de datos, ni cómo esto afectaría a la disponibilidad de *iQuality* ni a la durabilidad de los datos, ya que no se dispone de información de cómo ha sido implementado el Data mart.

Escalabilidad Por la misma razón, no podemos predecir cómo escalaría la aplicación si tuviese muchos usuarios concurrentes. Aunque esta situación no ocurre en la actualidad, podría ser importante tenerlo en cuenta en futuras versiones.

Seguridad Por falta de tiempo, no se tienen en cuenta aspectos de seguridad, y por lo tanto la aplicación es vulnerable a ataques tan básicos como XSS o SQL injection. En una futura versión, el tratamiento de la seguridad debería considerarse una prioridad, ya que están en juego datos muy sensibles.

Finalmente, es conveniente precisar que los procesos ejecutados en el Data mart quedan fuera del ámbito del presente proyecto, dado que actualmente no existe la necesidad de modificarlos. Tampoco se modificará la base de datos, pues consta de varios centenares de tablas[4], disparadores y vistas. Aunque sería posible optimizarlas en algunos aspectos, una refactorización de esta magnitud requeriría de varias semanas que simplemente no podemos invertir en un proyecto de tan corta duración.

3.2 Obstáculos y riesgos

Son muy variados e inesperados los obstáculos que pueden surgir cuando se acomete un proyecto de esta envergadura. Sin embargo, algunos son ciertamente predecibles y es muy importante identificarlos. En este caso, los obstáculos y riesgos que hay que superar son:

Entender el funcionamiento de *iQuality*. Se trata de una herramienta muy compleja y desarrollada por muchos ingenieros. Aunque este proyecto se centra en sólo una de sus facetas, resulta crucial tener una imagen global de cómo funciona y qué procesos la conforman para saber lo que se pretende conseguir con cada funcionalidad de la aplicación web.

Dominar el *framework* Spring. Aunque estoy bien familiarizado con Java y con el desarrollo de aplicaciones web, Spring es un *framework* que nunca he utilizado y del que no tengo ningún conocimiento. Llegar a desenvolverme con soltura en Spring resultará un aspecto básico

[4] 1794, para ser precisos.

para el éxito del proyecto. Será necesario invertir varias horas leyendo documentación y haciendo pruebas con algunos ejemplos sencillos antes de empezar con *iQuality*.

Gestionar el tiempo. Como consecuencia de los dos puntos anteriores, resulta difícil hacer una buena estimación del tiempo necesario para implementar cada funcionalidad cuando no se tiene un buen conocimiento ni de lo que se pretende implementar ni de la herramienta que se va a utilizar. Deberá efectuarse un recalibrado de la planificación temporal con cierta frecuencia para paliar estas incertidumbres.

3.3 Metodología y seguimiento

Dada la corta duración del proyecto y el hecho de que el director es un *Scrum master*, se ha decidido seguir las pautas de Scrum para gestionar el proyecto. Generaremos un *backlog* a partir de todas las funcionalidades que se quieren implementar. En una reunión con el director del proyecto y otros ingenieros que están involucrados en el mismo, priorizaremos, usando como medida tallas de camiseta, los elementos del *backlog* y estimaremos en puntos la dificultad de cada uno. Propondremos a continuación cuánto durarán los *sprints* y una velocidad de *sprint*, i. e., cuántos puntos se pueden implementar en el margen de tiempo acotado por un *sprint*.

Evidentemente, es probable que la velocidad no esté bien aproximada durante los primeros *sprints*, pues no disponemos de mucha información ni experiencia. Pero en función de cómo progrese cada *sprint*, la iremos ajustando paulativamente y no tardaremos en contar con una buena estimación.

Las reuniones se realizarán al final de cada *sprint*, cada uno de los cuales durará dos semanas. Estarán presentes el director del proyecto y algunos de los ingenieros que desarrollaron *iQuality*. Se tratarán los posibles problemas que hayan surgido durante la semana y se harán consultas, se comprobará si se han conseguido los objetivos establecidos, se repriorizará el *backlog* y re-estimará la velocidad si hiciese falta, y se propondrá el *sprint backlog* del siguiente *sprint*. Los objetivos se valorarán en estas reuniones mediante una *checklist* y unos criterios establecidos en el momento de plantearlos, y las funcionalidades serán validadas por los ingenieros de Indra que usan la versión actual de *iQuality*, los cuales verificarán si son capaces de realizar la acción correspondiente con la nueva versión y darán *feedback* de su experiencia como usuarios.

Una vez superados todos los objetivos planteados y los obstáculos que se presenten, dispondremos de una versión mejorada de *iQuality* que no sólo será más eficiente, más atractiva y más fácil de usar, sino que quedará documentada y preparada para que el próximo desarrollador o equipo pueda tomar el relevo e implementar la siguiente versión.

4 Planificación temporal

4.1 Calendario

El proyecto comienza oficialmente el 16 de septiembre de 2015, y finaliza el 26 de enero de 2016 con la defensa ante el tribunal. Esto significa que la documentación debe estar finalizada y entregada como máximo el día 19 de enero (i.e., una semana antes). Ahora bien, en vista de los posibles contratiempos que puedan surgir, se define como fecha límite el 31 de diciembre de 2015. Esto nos da casi tres semanas de margen en la planificación como medida preventiva en caso de problemas imprevistos. Habiendo fijado estas fechas, la duración del proyecto será de tres meses y medio.

La dedicación será de 35 horas por semana enfocados al desarrollo de *software* (y, en menor medida, a la planificación de las iteraciones) y alrededor de 5 o 10 horas por semana redactando la documentación. Esto nos da a una estimación de 40-45 horas por semana, llevadas a cabo por una sola persona.

4.2 Descripción de las fases

En cuanto a la planificación del proyecto podemos distinguir tres grandes bloques organizativos, que de describen en la siguientes subsecciones.

4.2.1 Planificación inicial

En esta fase definimos los requerimientos del proyecto, así como sus objetivos y los requisitos para satisfacerlos. Será durante esta etapa donde generemos la mayor parte de la documentación, en concreto la asociada a la asignatura de GEP: el documento de introducción y alcance, la planificación temporal, el estudio económico, etc.

Como ya se ha mencionado, la metodología que vamos seguir en cuanto a la gestión del proyecto será Scrum. Por lo tanto, este es el momento de elaborar el *backlog* con el director del proyecto. Será una simple lista con las funcionalidades y tareas que llevaremos a cabo, priorizadas y ponderadas en cuanto al tiempo estimado para completarlas. Evidentemente, la ponderación

será de carácter aproximado, pero se revisará el *backlog* tras cada iteración para ajustar las estimaciones y reponderar las prioridades. De esta forma disminuirá la incertidumbre de la planificación tras cada iteración.

Para priorizar las tareas utilizaremos una unidad de medida muy popular en las metodologías ágiles: las tallas de camiseta. Con esta forma de medir, una funcionalidad esencial tendrá una prioridad XXL, mientras que las de menor importancia tendrán una prioridad S. Evidentemente, podemos tener cualquier valor intermedio: M, L, XL. Téngase en cuenta que muchas de las tareas pueden no ser prioritarias, pero sí muy *necesarias* para continuar con el proyecto. Por ejemplo, preparar el entorno de trabajo no es uno de los objetivos fundamentales, pero si no lo hacemos no podremos implementar nada en absoluto.

En cuanto al coste temporal de una tarea, se estimará con una puntuación que puede tomar los valores 1, 2, 3, 5 u 8. Lo más importante son los ordenes de magnitud relativos, e.g., una tarea con coste 2 requerirá el doble de tiempo que una de coste 1. Cabe destacar también que los valores son adimensionales: una tarea con coste 3 tardará en llevarse a cabo aproximadamente el triple de tiempo que una tarea de coste 1, pero en general esto no significa que se tardará 3 horas o 3 días.

Dicho esto, he aquí el *backlog* del proyecto (sin ningún orden en particular):

Cuadro 4.1: *Backlog* inicial del proyecto

Tarea	**Prioridad**	**Coste**
Configuración del entorno de trabajo: STS, Maven, SQLDeveloper y Git.	S	1
Probar ejemplos de proyectos simples en Spring y leer documentación del *framework*.	S	1
Implementar el esqueleto del *backend* de la aplicación: conexión a la base de datos y mostrar datos por una vista sencilla.	M	3
Implementar el esqueleto del *frontend* de la aplicación: logo, menú, usuario, desplegable, etc.	M	3
Analizar la interfaz y funcionalidades del *iQuality* actual; identificar elementos visuales e interacciones más importantes.	S	1
Investigar sobre *templates* y *frameworks* de *frontend* y elegir uno.	S	2
Instalar el *template frontend* e integrarlo al esqueleto del proyecto.	M	2
Continúa en la página siguiente		

Cuadro 4.1 – continuación de la página anterior

Tarea	Prioridad	Coste
Funcionalidad de consulta del estado de las ejecuciones lanzadas y cada uno de los jobs que las componen.	XXL	5
Funcionalidad de consulta de las certificaciones disponibles.	XXL	3
Funcionalidad de consulta del resultado de las certificaciones a nivel global y a nivel detallado individual.	XXL	5
Funcionalidad de alta, baja o modificación de certificaciones.	XL	3
Funcionalidad de consulta del árbol de entidades, dimensiones e indicadores.	XXL	8
Funcionalidad de consulta de las fichas de definición de cada elemento terminal del árbol (atributos e indicadores).	XXL	8
Tomar decisiones de diseño del *frontend*: iconos, tipografía, colores, etc.	M	1
Refactorizar y comentar el código; generar Javadoc.	XL	5
Exportar datos a Excel.	XL	2
Importar datos desde Excel.	XXL	3
Funcionalidad de *login* y gestión de contraseñas.	M	3
Gestión de sesión para no tener que hacer *login* constantemente.	S	1
Funcionalidad de administración de perfiles y usuarios.	M	3
Funcionalidad (nueva) de alerta cuando una ejecución ha fallado.	L	8
Funcionalidad (nueva) que muestre un tablero con métricas y gráficas de la calidad de los datos del sistema.	XL	8
Funcionalidad (nueva) que permita parametrizar las métricas de calidad de los datos.	L	5
Funcionalidad (nueva) para lanzar las ejecuciones desde la aplicación.	L	5

Continúa en la página siguiente

Cuadro 4.1 – continuación de la página anterior

Tarea	Prioridad	Coste
Gestión de escenarios.	M	3
Redactar la memoria.	XXL	8

A partir de este *backlog*, decidimos qué tareas podemos llevar a cabo dentro del alcance temporal del proyecto, y en qué orden. Esto corresponderá a los diferentes *sprint*s, cada uno con su *sprint backlog* (elaboraremos sobre los detalles de cada *sprint* en la Subsección 4.2.2). Se ha acordado con el director del proyecto que la duración de un *sprint* será de 2 semanas (laborables). Por lo tanto, disponemos de 6 *sprint*s, más uno extra de contingencia durante la primera mitad de enero. Parte de este último servirá a su vez para las tareas de documentación y refactorización, como veremos en la Subsección 4.2.3.

A continuación debemos decidir una velocidad de *sprint*; consideraremos que tenemos la capacidad de completar entre 8 y 10 puntos en un *sprint*, y ajustaremos este valor a medida que ganemos experiencia en el proyecto.

En el *backlog* se puede observar que un aspecto importante de la planificación inicial consiste en tomar decisiones respecto a las tecnologías que hay que utilizar, así como familiarizarnos con ellas. Estas no son funcionalidades que queremos ofrecer mediante la aplicación, pero debemos darles importancia ya que son el fundamento del proyecto. Las tecnologías utilizadas, así como los motivos que han determinado su elección, se describen en la Sección 8.3.

4.2.2 Fase de desarrollo

En esta fase analizamos, diseñamos e implementamos las funcionalidades que queremos ofrecer. Al mismo tiempo, generamos la documentación de carácter más técnico del proyecto: diagramas de clases, esquemas y bocetos de la interfaz, Javadoc, etc.

El desarrollo está dividido en *sprint*s, con la idea de que al final de cada *sprint* tengamos una versión operativa de la herramienta, pero con un subconjunto de las funcionalidades del producto final. Además, cada *sprint* es incremental, con lo cual tras cada iteración tendremos una versión más completa del producto. Al final de cada *sprint* se llevará a cabo una reunión con el director y otro personal relacionado con el proyecto para realizar una demostración de las funcionalidades implementadas y planificar el próximo *sprint*.

Ahora que ya sabemos cómo se estructuran los *sprint*s y tenemos un *backlog* priorizado y ponderado, podemos proponer cuáles serán los sprints que deberíamos ser capaces de llevar a cabo dentro de los márgenes temporales del proyecto:

Sprint 1 Coste estimado: 9 puntos.

- Configuración del entorno de trabajo: STS, Maven, SQLDeveloper y Git.
- Probar ejemplos de proyectos simples en Spring y leer documentación del *framework*.
- Implementar el esqueleto del *backend* de la aplicación: conexión a la base de datos y mostrar datos por una vista sencilla.
- Implementar el esqueleto del *frontend* de la aplicación: logo, menú, usuario, desplegable, etc.
- Analizar la interfaz y funcionalidades del *iQuality* actual; identificar elementos visuales e interacciones más importantes.

Sprint 2 Coste estimado: 9 puntos.

- Investigar sobre *template*s y *framework*s de *frontend* y elegir uno.
- Instalar el *template frontend* e integrarlo al esqueleto del proyecto.
- Funcionalidad de consulta del estado de las ejecuciones lanzadas y cada uno de los jobs que las componen.

Sprint 3 Coste estimado: 8 puntos.

- Funcionalidad de consulta del árbol de entidades, dimensiones e indicadores.

Sprint 4 Coste estimado: 8 puntos.

- Funcionalidad de consulta de las fichas de definición de cada elemento terminal del árbol (atributos e indicadores).

Sprint 5 Coste estimado: 8 puntos.

- Funcionalidad de consulta de las certificaciones disponibles.
- Funcionalidad de consulta del resultado de las certificaciones a nivel global y a nivel detallado individual.

Sprint 6 Coste estimado: 8 puntos.

- Importar datos desde Excel.
- Exportar datos a Excel.
- Funcionalidad de administración de perfiles y usuarios.

4.2.3 Fase final

Durante la etapa final del proyecto recopilaremos en un único volumen toda la documentación generada, y nos cuidaremos bien de revisarla y corregir cualquier imperfecto que pudiésemos encontrar. Para esta tarea será muy importante el *feedback* recibido tanto del director como del ponente. Asimismo, deberemos poner especial atención a las posibles incoherencias que podamos encontrar en el documento a causa de los cambios de planificación que son naturales en cualquier proyecto de *software*.

Como se ha visto en la Subsección 4.2.2, lo más probable es que los tres meses y medio de duración del proyecto no sean suficientes para obtener un producto plenamente acabado, sino una primera versión que cubra las funcionalidades y aspectos más importantes y fundamentales. Es por eso que, en esta fase, será conveniente dejar una lista de las tareas que faltan por hacer y de ideas para una futura versión de *iQuality*. Junto con el presente documento y el Javadoc, facilitarán el trabajo a la persona o equipo que deba tomar el relevo del proyecto.

Por lo tanto, el *backlog* del *sprint* final constará de:

Sprint 7 Coste estimado: 13 puntos.[1]

- Refactorizar y comentar el código; generar Javadoc.
- Redactar la memoria.
- Redactar lista de tareas y aspectos a tener en cuenta para los próximos desarrolladores.

Finalmente, es en esta etapa donde se preparará la defensa del proyecto ante el tribunal. Es un buen momento para hablar con el ponente acerca de todos los aspectos de la defensa, así como ensayar e informarse al máximo sobre qué esperar el día que nos encontremos ante el tribunal.

[1] Aunque en el *backlog* en la Subsección 4.2.1 hemos dado un coste de 8 a la documentación, lo cierto es que la redactaremos a lo largo de todo el proyecto, con la intención de que para el últimos *sprint* queden pocos apartados por redactar. Por eso podemos esperar que el coste real de este último *sprint* sea mucho menor que 13 puntos.

4.3 Valoración de alternativas y plan de acción

Somos conscientes de que puede haber ligeras desviaciones en la duración de los diferentes *sprint*s. En este sentido, consideramos que la elección de una metodología ágil resultará muy apropiada. Realizaremos reuniones de seguimiento al final de cada *sprint*, donde se revaluará la planificación acorde al estado actual del proyecto y se realizarán los ajustes convenientes.

Si detectamos que una tarea está retrasando significativamente la realización de las siguientes, se reajustará la planificación. En caso de que el retraso empiece a ser preocupante (del orden de un *sprint*), podremos ponernos en contacto con alguno de los desarrolladores de Indra para pedir ayuda y resolver las dificultades. Por otra parte, téngase en cuenta la precaución que hemos tenido al dejar la mayor parte del último *sprint* como margen antes de la entrega, lo cual nos proporciona cierta flexibilidad en caso de contingencias. Gracias a estos dos aspectos reducimos enormemente las probabilidades de un desfase temporal importante en la realización del proyecto.

Por el contrario, si nos diésemos cuenta de que algunas tareas las estamos realizando más rápidamente de lo esperado, en las reuniones de seguimiento podremos redefinir los *sprint*s para implementar funcionalidades nuevas.

4.4 Recursos

En la realización de cualquier proyecto es muy importante tener presente desde el principio los recursos necesarios para su desarrollo. En el presente proyecto, los recursos serán tanto humanos como materiales.

4.4.1 Recursos humanos

Identificaremos los roles típicos de la mayoría de los proyectos software: jefe de proyecto, analista, diseñador, desarrollador y tester. Aun así, al tratarse de un proyecto de final de grado, los cuatro últimos roles serán asumidos por el autor del proyecto, mientras que el jefe o director será una persona responsable de Indra. Sin embargo, al estar en ocasiones involucrado de manera más apremiante en otros proyectos, es posible que el autor realice también, esporádicamente, tareas propias del jefe de proyecto.

Jefe de proyecto Encargado de planificar, organizar, coordinar, controlar y liderar el proyecto. Será Marc Ruiz Figueras, gerente BI de Indra.

Analista Encargado de analizar los requisitos del proyecto, concretándolos mediante el contacto

con el cliente. El presente proyecto se basa en uno anterior, y los requisitos serán prácticamente los mismos. Por lo tanto el contacto con el cliente no será una parte importante.

Diseñador Encargado de decidir el diseño de la interfaz de la aplicación así como su arquitectura interna, aplicando los patrones de diseño apropiados.

Programador Encargado de implementar la aplicación a partir de los requisitos y el diseño.

Tester Encargado de realizar las pruebas del sistema y comprobar que las diversas funcionalidades se ejecutan acorde a los requerimientos y especificaciones.

4.4.2 Recursos materiales

Son los recursos que los diferentes roles del proyecto necesitan para llevar a cabo sus tareas. Identificamos los siguiente:

- Ordenador portátil HP donde se llevarán a cabo las tareas de diseño, desarrollo y *testing*.
- Microsoft Office Enterprise Edition para organizar el *backlog* en hojas de cálculo.
- LaTeX para generar la presente documentación; es gratuito.
- STS: IDE que facilita mucho el uso de Spring y la gestión de dependencias con maven; es gratuito.
- Maven: para gestionar automáticamente las dependencias del proyecto; es gratuito.
- Apache Tomcat: para realizar pruebas de la aplicación en un entorno local; es gratuito.
- GitHub: para gestionar las versiones del código y tener una copia de seguridad. Una licencia para estudiantes nos permite tener un repositorio privado gratuito.
- Data mart: es una sección del centro de procesamiento de datos de Indra dedicada al proyecto *iQuality*.
- Electricidad: tanto para desarrollar el proyecto como para alimentar el Data mart.
- *template frontend*: plantilla que nos ahorrará mucho trabajo y tiempo en el diseño de la interfaz gráfica de la aplicación.

4.5 Diagrama de Gantt

Para poder visualizar más fácilmente la planificación arriba expuesta, a continuación presentamos el diagrama de Gantt que resume todas las tareas y el orden planificado en el que se llevarán a cabo, así como las principales dependencias entre tareas.

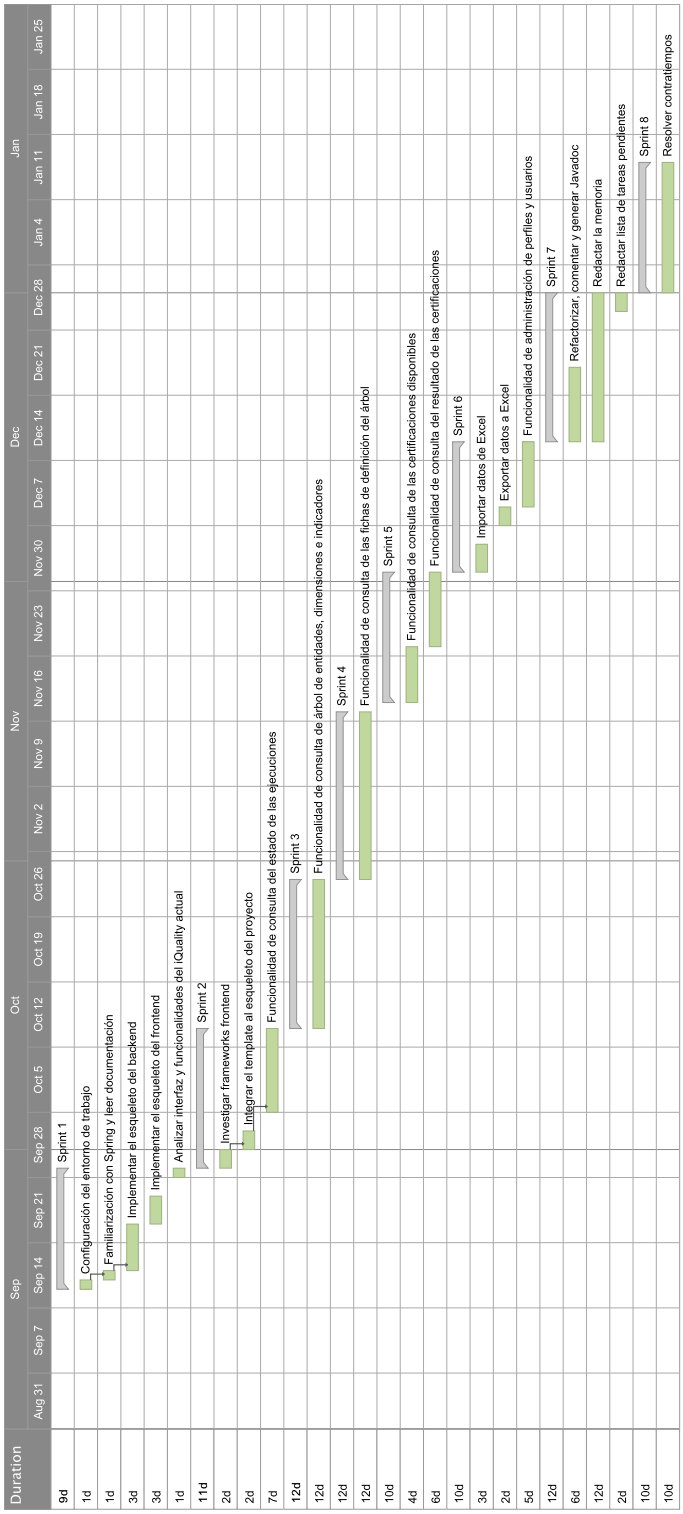

Figura 4.1: Diagrama de Gantt de las tareas a realizar en el proyecto.

5 Gestión económica

5.1 Estimación de costes

Una vez identificados los recursos necesarios para el proyecto, podemos proceder a estudiar y estimar los costes directos e indirectos asociados al mismo. Esto nos permitirá evaluar la viabilidad económica del proyecto y tomar la decisión de seguir adelante con el plan previsto o ajustar el uso de recursos antes de continuar.

5.1.1 Costes en recursos humanos

El salario del jefe de proyecto se pueden estimar en torno a los 20€/h, mientras que el del becario viene dado por el convenio entre la FIB e Indra, y es de 8€/h. A partir de las tareas mostradas en el diagrama de Gantt de la Sección 4.5, podemos ver cuántos días serán necesarios para cada tarea del *backlog*. Teniendo en cuenta que un día laboral es una jornada de 5 horas y todas las tareas del *backlog* las realiza el becario, obtenemos el coste monetario en cuanto a recursos humanos de cada tarea:

Cuadro 5.1: Coste de las tareas del *backlog*

Tarea	**Horas**	**Coste**
Configuración del entorno de trabajo: STS, Maven, SQLDeveloper y Git	5	40 €
Probar ejemplos de proyectos simples en Spring y leer documentación del *framework*	5	40 €
Implementar el esqueleto del *backend* de la aplicación: conexión a la base de datos y mostrar datos por una vista sencilla	15	120 €
Implementar el esqueleto del *frontend* de la aplicación: logo, menú, usuario, desplegable, etc.	15	120 €
Continúa en la página siguiente		

Cuadro 5.1 – continuación de la página anterior

Tarea	Horas	Coste
Analizar interfaz y funcionalidades del *iQuality* actual; identificar elementos visuales e interacciones más importantes	5	40 €
Investigar sobre *template*s y *framework*s de *frontend* y elegir uno	10	80 €
Instalar el *template frontend* e integrarlo al esqueleto del proyecto	10	80 €
Funcionalidad de consulta del estado de las ejecuciones lanzadas y cada uno de los jobs que las componen	35	280 €
Funcionalidad de consulta de árbol de entidades, dimensiones e indicadores	60	480 €
Funcionalidad de consulta de las fichas de definición de cada elemento terminal del árbol (atributos e indicadores)	60	480 €
Funcionalidad de consulta de las certificaciones disponibles	20	160 €
Funcionalidad de consulta del resultado de las certificaciones a nivel global y a nivel detallado individual	30	240 €
Importar a desde Excel	15	120 €
Exportar datos a Excel	10	80 €
Funcionalidad de administración de perfiles y usuarios	25	200 €
Refactorizar y comentar el código; generar Javadoc	30	240 €
Redactar la memoria	60	480 €
Redactar lista de tareas pendientes	10	80 €
Total		**3360 €**

Además, por cada *sprint* se realizará una reunión de aproximadamente una hora con el director. Tenemos 7 *sprint*s, pero lo redondearemos a 10 horas para tener en cuenta tiempo de consultas, con lo cual tenemos un coste adicional de 200 € por el tiempo del director. Sumados a los 3360 € del becario, el coste total en recursos humanos es de 3560 € $\simeq$ 4000 €.

5.1.2 Costes en recursos materiales

El coste de los recursos materiales descritos en la Subsección 4.4.2, sin incluir los que ya se ha indicado que son gratuitos, viene dado por:

1. Costes directos

   - Ordenador portátil HP EliteBook 840: lo proporciona la empresa y ya ha sido amortizado. Podemos considerar que lo obtenemos por coste 0.

   - Microsoft Office Enterprise Edition: lo proporciona la empresa y ya ha sido amortizado. Podemos considerar que lo obtenemos por coste 0.

   - *Template frontend*: actualmente estamos usando una licencia no comercial por valor de 24 €. Si finalmente se vende el producto, se debe tener en cuenta que la licencia comercial asciende a 1200 €.

2. Costes indirectos

   - Electricidad para el desarrollo: volviendo al diagrama de Gantt, pasamos los días dedicados por el becario a horas. Teniendo en cuenta que el consumo energético del portátil es de 65 W [5] y el coste de la energía contratada para empresas en hora punta es de 0,169682 €/kWh [17] tenemos que:

   $$86\,\text{día} \cdot 5\,\frac{\text{h}}{\text{día}} \cdot 65 \times 10^{-3}\,\text{kW} \cdot 0.169\,682\,\frac{€}{\text{kW h}} = 4742.6119\,€ \tag{5.1}$$

   - Electricidad y mantenimiento Data mart: está activo todo el año y forma parte de un Centro de Procesamiento de Datos (CPD) utilizado para varios proyectos. Estimamos que la parte del CPD dedicada a *iQuality* es de un 2 %. Teniendo en cuenta que el consumo energético promedio de un CPD es del orden de 1000 kW [14], tenemos que:

   $$0{,}02 \cdot 24\,\frac{\text{h}}{\text{día}} \cdot 365\,\frac{\text{día}}{\text{año}} \cdot 1000\,\text{kW} \cdot 0.169\,682\,\frac{€}{\text{kW h}} = 29\,728.2864\,\frac{€}{\text{año}} \tag{5.2}$$

   Por otro lado, estimamos que el CPD tiene un coste de mantenimiento anual del orden de 20000 €, por lo tanto debemos añadirlo al coste anterior:

   $$0{,}02 \cdot 20\,000\,\frac{€}{\text{año}} = 4000\,\frac{€}{\text{año}} \tag{5.3}$$

   En resumen, tenemos un coste total en recursos materiales de: 5942,6119 € fijos más 33728,2864 € anuales $\simeq$ 6000 € fijos más 35000 € anuales.

3. Coste total

Teniendo en cuenta todos los costes arriba descritos, el coste total del proyecto será de 10000 € fijos, más un mantenimiento anual de 35000 €.

Ahora bien, como plan de contingencia contra posibles desviaciones en el presupuesto, contemplaremos un margen de un 20 % superior al estimado, con lo cual el coste total del proyecto realmente ascenderá a 12000 € fijos, más un mantenimiento anual de 42000 €.

5.2 Control de gestión

Para realizar el control de gestión, se mantendrá un historial de los recursos humanos y no humanos consumidos.

Para los recursos humanos, cada día anotaremos en una tabla las horas de trabajo realizadas y en qué tareas se han invertido. De esta forma, al finalizar el proyecto podremos comparar las horas invertidas con las estimadas y calcular la desviación en presupuesto debida a los sueldos de los integrantes del equipo.

En cuanto a los recursos no humanos, algunos tienen un coste fijo y bien definido (como las licencias), y otros pueden variar con el tiempo, en particular el mantenimiento y consumo del Data mart. Este coste se registrará cada mes y se comprobará que se ajuste a los costes estimados. Si descubrimos grandes desviaciones nos pondremos en contacto con el servicio técnico del Data mart para descubrir las anomalías, y si la utilización del mismo es más intensa que la prevista se realizarán los ajustes de presupuesto a largo plazo que resulten más apropiados.

De esta manera, las posibles desviaciones en el presupuesto a causa del consumo de recursos serán registradas, y en la etapa de finalización y seguimiento se podrán analizar sus causas. Esta información resultará útil como base de experiencia para la planificación de futuros proyectos de características similares.

6 Sostenibilidad y compromiso social

En el desarrollo de cualquier proyecto, y en particular en un proyecto técnico, la ética profesional obliga a tener en cuenta el impacto que tendrá sobre el entorno. En concreto, debemos tener en cuenta la sostenibilidad económica, social y ambiental del proyecto y de los artefactos que se generarán.

6.1 Sostenibilidad económica

Como se ha visto en más detalle en el Capítulo 5, al tratarse de un proyecto reducido y a ser desarrollado por una sola persona, no se precisa gran cantidad de recursos materiales ni humanos. Incluso teniendo en cuenta los riesgos principales, disponemos de un buen margen de acción que hace del proyecto una buena apuesta en cuanto a viabilidad económica.

Al no requerir muchos recursos, es a a su vez un proyecto competitivo, ya que aunque redujésemos el margen de ganancias por su comercialización, seguiría siendo rentable.

Además, no se debe perder de vista la gran ventaja económica de este proyecto: si se completa con éxito, no sólo será un producto para VidaCaixa, sino que será muy fácil y barato adaptarlo para otros clientes con necesidades similares. Esto significa que *iQuality* podrá ser comercializado con un margen de ganancias mucho mayor a partir del segundo cliente.

Por otra parte, se ha hecho más hincapié en los aspectos técnicos y específicos del proyecto, intentando dedicar los mínimos esfuerzos necesarios a aspectos que se puedan reaprovechar de otras fuentes. Un buen ejemplo de esto es la compra del *template* para el *frontend*. El desarrollo de un *frontend* desde cero hubiese requerido una cantidad de horas que no era necesario invertir.

6.2 Sostenibilidad social

El proyecto mejora la calidad de los datos de VidaCaixa, por lo tanto los empleados de esta entidad podrán realizar informes mejores y menos costosos. Esto afecta, de manera indirecta, a los clientes de VidaCaixa, ya que si los directivos conocen mejor su negocio les podrán ofrecer

un mejor servicio. Evidentemente, esto es cierto siempre que se opere dentro de unos límites éticos asumidos.

Aun así, la mejora en el servicio hacia los usuarios puede llegar a ser poco significativa en comparación con la mejora que supone para los empleados de VidaCaixa. *iQuality* cubre una necesidad real de cualquier gran empresa que manipule una gran cantidad de datos, y por lo tanto afecta positivamente a los empleados al facilitarles el trabajo.

Por otro lado, el proyecto almacenará y manipulará datos delicados de diversas organizaciones, por lo que es muy importante que cumpla la normativa establecida por la Ley Orgánica de Protección de Datos (LOPD). Aunque el tratamiento de los datos en el Data mart queda fuera del alcance del proyecto, deberá tenerse en cuenta en futuras etapas del mismo.

Finalmente, *iQuality* facilita la generación de los informes periódicos de las actividades de las entidades financieras. Estos informes son requeridos por entidades legales reguladoras a nivel europeo, acorde a la normativa EIOPA. Por lo tanto, el proyecto favorecerá el cumplimiento de las leyes que se ven involucradas en las solicitud de estos informes, y deberá calcular los indicadores siguiendo estrictamente la normativa prescrita por la EIOPA.

6.3 Sostenibilidad ambiental

Los recursos necesarios son casi en su totalidad energéticos, en concreto electricidad. No se requieren materias primas ni recursos manufacturados, aparte de las herramientas de software antes mencionadas. Parte de la electricidad sólo será necesaria durante los pocos meses de duración del desarrollo, y tiene un impacto ambiental insignificante.

La otra parte de los recursos energéticos serán consumidos por el Data mart durante la vida útil de la aplicación. Sin embargo, cabe decir que el Data mart forma parte de un CPD que lleva en funcionamiento varios años, y por lo tanto su utilización no supone ningún coste adicional que no se tuviese sin el proyecto.

Finalmente, se reducirá el consumo de papel ya que será mucho más fácil, interactivo e informativo acceder a los informes a través de la aplicación que a través de impresiones en papel.

6.4 Tabla de sostenibilidad

Teniendo en cuenta los aspectos arriba mencionado, hemos elaborado la siguiente tabla de sostenibilidad, donde la valoración es entre 0 y 10:

Sostenibilidad	Económica	Social	Ambiental
Valoración	7	6	7

7 Especificación

7.1 Requisitos funcionales

Como se ha visto en la Sección 3.3, la metodología usada para el desarrollo del proyecto es Scrum. Por lo tanto la toma de requisitos fue en forma de historias de usuario durante las reuniones de seguimiento con el director y otros miembros implicados en la versión anterior de *iQuality*. En este caso eran ellos quienes jugaban el papel de clientes, ya que determinaban el resultado esperado al migrar la aplicación.

Las historias de usuario siguen el siguiente formato:

#0 Historia de usuario
Como usuario de algún tipo **quiero** hacer algo **para** obtener un resultado
Criterios de aceptación
• Un criterio • Otro criterio

Se escriben en lenguaje natural, y expresan una funcionalidad (*quiero*) que un usuario (*como*) espera obtener del sistema para satisfacer una necesidad (*para*). Cada historia de usuario debe cumplir ciertos criterios de aceptación definidos por los cliente para que la historia sea considerada como finalizada y aceptada.

Dicho esto, tenemos a continuación, las historias de usuario definidas para el proyecto:

#1 Consultar ejecuciones
Como usuario de validación-funcional **quiero** ver un registro de las ejecuciones **para** poder comprobar el estado de un proceso
Criterios de aceptación
• Para cada ejecución, puedo ver las dimensiones: identificador, pase, estado, fecha de inicio, fecha de fin y duración. • Puedo ordenar las ejecuciones por cualquiera de las dimensiones. • Puedo buscar ejecuciones por cualquier dimensión escribiendo texto. • Puedo limitar el número de ejecuciones mostradas. • Puedo ver las ejecuciones para cada versión de software disponible. • Puedo seguir un enlace al detalle de la ejecución.

#2 Consultar jobs
Como usuario de validación-funcional **quiero** ver un registro de los jobs de una ejecución **para** saber el estado e información de cada job
Criterios de aceptación
• Los jobs se agrupan según su ejecución • Para cada job, puedo ver las dimensiones: identificador, estado, fecha de inicio, fecha de fin, si es un punto de control y la duración. • Puedo ordenar los jobs por cualquiera de las dimensiones. • Puedo buscar jobs por cualquier dimensión escribiendo texto. • Puedo limitar el número de jobs mostradas. • Puedo visualizar las dependencias de cada job. • Puedo seguir un enlace al registro de operaciones del job.

#3 Consultar registro de operaciones

Como usuario de validación-funcional

quiero ver el registro de operaciones de un job

para analizar detalladamente las acciones realizadas por cada job.

Criterios de aceptación

- Los registros de operaciones se agrupan según su job en una ejecución.
- Para cada registro, puedo ver las dimensiones: identificador, fecha de inicio, duración, tipo de operación, estado, y filas actualizadas/cargadas/leídas/rechazadas/descartadas de tablas.
- Puedo ordenar los registros por cualquiera de las dimensiones.
- Puedo buscar registros por cualquier dimensión escribiendo texto.
- Puedo limitar el número de registros mostradas.
- Puedo visualizar el *log* dejado por cada operación.

#4 Consulta de pases

Como usuario de validación-funcional

quiero ver un registro de los pases definidos

para conocer los pases que puedo usar en el sistema.

Criterios de aceptación

- Para cada pase, puedo ver las dimensiones: software, identificador y nombre.
- Puedo ordenar los pases por cualquiera de las dimensiones.
- Puedo buscar pases por cualquier dimensión escribiendo texto.
- Puedo limitar el número de pases mostrados.

#5 Crear nuevo pase

Como usuario funcional

quiero crear un nuevo pase

para poder lanzar ejecuciones con nuevos comportamientos.

Criterios de aceptación

- Puedo introducir el nombre del pase. El sistema no me permite usar un nombre repetido o con menos de 4 letras.
- Seleccionar fácilmente los jobs que quiero asociar al pase. El sistema me obliga a seleccionar al menos un job.
- Crear dependencias entre los jobs anteriores.
- Si no se cumple alguna de las condiciones mencionadas, el sistema me avisa y no me deja crear el pase.
- Si se cumplen las condiciones, el sistema me avisa que ha creado el pase y me muestra todos los pases.

#6 Consultar certificaciones de negocio

Como usuario de validación

quiero ver las certificaciones de negocio

para conocer las reglas de negocio definidas en el sistema.

Criterios de aceptación

- Para cada certificación de negocio, puedo ver las dimensiones: fecha, sección, subsección, entidad, nombre, indicador y estado.
- Puedo ordenar las certificaciones por cualquiera de las dimensiones.
- Puedo buscar certificaciones por cualquier dimensión escribiendo texto.
- Puedo limitar el número de certificaciones mostradas.
- Puedo seguir un enlace a la vista detallada de la certificación.

#7 Consultar detalle de certificación de negocio

Como usuario de validación

quiero ver los detalles de una certificaciones de negocio

para conocer el estado de las reglas de negocio aplicadas sobre entidades del sistema.

Criterios de aceptación

- Para la certificación de negocio, puedo ver las dimensiones características para cada entidad sobre la que se ha aplicado.
- Además puedo ver la fecha, indicador y nombre de la certificación.
- Puedo ordenar los detalles por cualquiera de las dimensiones.
- Puedo buscar detalles por cualquier dimensión escribiendo texto.
- Puedo limitar el número de detalles mostrados.

#8 Consultar validaciones técnicas

Como usuario de validación

quiero ver las validaciones técnicas

para conocer las reglas técnicas definidas en el sistema.

Criterios de aceptación

- Para cada validaciones técnicas, puedo ver las dimensiones: fecha, sección, subsección, entidad, nombre, registros y estado.
- Puedo ordenar las validaciones por cualquiera de las dimensiones.
- Puedo buscar validaciones por cualquier dimensión escribiendo texto.
- Puedo limitar el número de validaciones mostradas.
- Puedo seguir un enlace a la vista detallada de la validación.

#9 Consultar detalle de validación técnica

Como usuario de validación

quiero ver los detalles de una validación técnica

para conocer el estado de las reglas técnicas aplicadas sobre entidades del sistema.

Criterios de aceptación

- Para la validación técnica, puedo ver las dimensiones características para cada entidad sobre la que se ha aplicado.
- Además puedo ver la fecha, entidad validada y nombre de la validación.
- Puedo ordenar los detalles por cualquiera de las dimensiones.
- Puedo buscar detalles por cualquier dimensión escribiendo texto.
- Puedo limitar el número de detalles mostrados.

#10 Crear nueva certificación

Como usuario funcional

quiero crear una nueva certificación de negocio o técnica

para poder parametrizar las medidas de la calidad de los datos.

Criterios de aceptación

- Puedo introducir el nombre de la certificación. El sistema no me permite usar un nombre repetido o con menos de 4 letras.
- Puedo seleccionar el tipo de la certificación.
- Puedo introducir únicamente los datos propios del tipo de la certificación.
- Si no se cumple alguna de las condiciones mencionadas, el sistema me avisa y no me deja crear la certificación.
- Si se cumplen las condiciones, el sistema me avisa que ha creado la certificación y me muestra todas las certificaciones.

#11 Cargar ficheros Excel

Como usuario de validación

quiero cargar ficheros Excel

para introducir tablas en el sistema de forma sencilla.

Criterios de aceptación

- Puedo seleccionar un fichero Excel mediante un navegador de directorios o arrastrando un fichero.
- Puedo seleccionar la tabla asociada al fichero.
- El sistema me informa cuando el fichero ha sido cargado y procesado.

#12 Consulta del diccionario de conceptos

Como usuario de validación

quiero visualizar los conceptos del negocio

para tener información sobre los conceptos usados.

Criterios de aceptación

- Los conceptos se muestran anidados en la jerarquía correspondiente.
- Los diferentes tipos de conceptos se representan con iconos distintivos diferentes.
- Al seleccionar un concepto del tipo indicador o atributo, tengo una vista detallada.

#13 Consultar detalles de un concepto del diccionario.

Como usuario de validación

quiero ver los detalles asociados a un concepto del tipo atributo o indicador

para tener información sobre los conceptos del negocio.

Criterios de aceptación

- Si el concepto es de tipo indicador, puedo ver: definición, comentarios, unidad de medida, periodo acumulado, responsable, método de obtención, histórico y las reglas de certificación que utiliza.
- Si el concepto es de tipo atributo, puedo ver: definición, comentarios, formato, responsable, método de obtención, trazabilidad, histórico y características de actualización.
- Además, si el atributo tiene una tabla maestra, puedo ver método de obtención, trazabilidad, histórico y características de actualización del maestro.

#14 Alertas ejecuciones KO
Como usuario funcional **quiero** recibir una alerta cuando una ejecución ha fallado (estado KO) **para** buscar la causa del fallo y volver a lanzar la ejecución.
Criterios de aceptación
• Veo un icono destacado cuando una ejecución falla. • Puedo seguir un enlace en el icono hasta la ejecución que ha fallado.

7.2 Requisitos no funcionales

Estos son los requisitos que no definen el comportamiento del sistema, sino sus características menos tangibles, pero no por ello menos importantes. Vienen descritas por una definición con parámetros que permiten medir si el requisito se cumple.

#1 Flexibilidad
Añadir una nueva pantalla con funcionalidades no complejas debe requerir menos de 5 horas de desarrollo.

#2 Portabilidad
La aplicación debe ser fácilmente instalable y funcionar en cualquier distribución moderna de Windows (7 en adelante) o Ubuntu (12 en adelante).

#3 Rendimiento
El diccionario de conceptos debe tardar menos de 4 segundos en cargarse.

#4 Rendimiento

Los buscadores de textos en las tablas deben responder mientras se teclea con una demora de menos de 1 segundo.

#5 Reusabilidad

Más de la mitad de los módulos y clases del *backend* de la aplicación deben poder reaprovecharse para aplicaciones similares en diferentes clientes. Esto no incluye la capa de datos.

#6 Reusabilidad

El *frontend* de la aplicación debe poder adaptarse a otros colores y distribuciones sin modificar mucho código.

#7 Robustez

La aplicación atrapará errores al acceder a la base de datos y redirigirá al usuario a una página de error.

#8 Usabilidad

La aplicación deberá ser intuitiva y autoexplicativa para un usuario que esté acostumbrado a la versión anterior.

#9 Trazabilidad

La aplicación deberá dejar un *log* de las operaciones y métodos llamados para que los desarrolladores puedan seguir el hilo de lo que hace el *software*.

#10 Mantenibilidad

Las partes críticas del código deben quedar comentadas y se dispondrá de una documentación de consulta sobre los módulos de la aplicación.

#11 Cambiabilidad

Para adaptar la aplicación a otros modelos de datos, sólo será necesario modificar el código asociado al acceso de datos.

8 Arquitectura y diseño

Para el diseño de cualquier producto *software* es muy importante utilizar los patrones arquitectónicos adecuados. Esto es especialmente cierto para proyectos de gran envergadura, como el que estamos analizando.

Un patrón de diseño es una abstracción genérica de una solución a un problema habitual. Esta solución se puede implementar de manera específica para resolver problemas conocidos de forma sencilla. Como son soluciones probadas y ampliamente utilizadas, su eficacia está garantizada. Además, están bien documentadas y existe mucha información sobre ellas, lo cual reduce notablemente los riesgos de un proyecto. Describiremos los patrones de diseño y arquitectónicos que hemos considerado necesarios para el proyecto en la Sección 8.1 y Sección 8.2.

Por otra parte, hemos seguido el clásico diseño de tres capas: capa de presentación (la que se presenta al usuario), capa de dominio (la lógica de negocio de la aplicación) y capa de datos (la persistencia física de la aplicación), como veremos en la Sección 8.4, Sección 8.5 y Sección 8.6.

8.1 Patrones arquitectónicos

8.1.1 MVC

Como ya se ha descrito al hablar sobre la implementación anterior de *iQuality*, la lógica de la aplicación estaba imbricada con el acceso a datos y proceso de visualización. Desde el punto de vista de la arquitectura de *software*, esto es poco menos que un pecado. Para paliar este problema, usaremos el modelo MVC, que divide los elementos de la aplicación en:

Modelo La representación lógica de los objetos.

Vista La forma en que los objetos se presentan al usuario.

Controlador La interacción entre la representación en la vista y los objetos del modelo.

Esto se puede resumir en el siguiente diagrama:

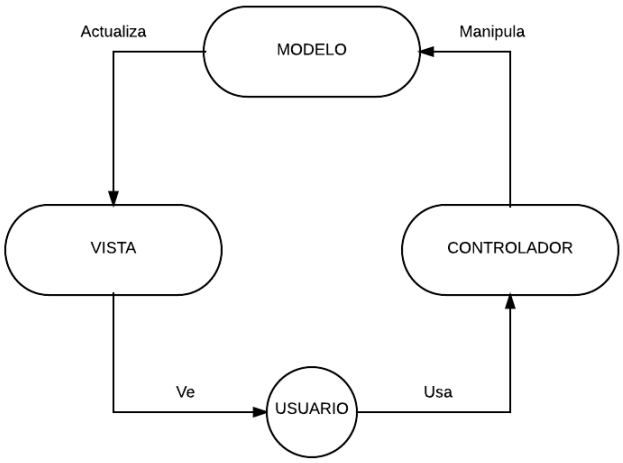

Figura 8.1: Diagrama de flujo de una aplicación MVC.

8.1.2 API REST

REpresentational State Transfer (REST) es la arquitectura subyacente a la Web. Define el conjunto de convenciones y restricciones que hacen de la Web un sistema práctico, eficiente y, en definitiva, funcional. Por este motivo, es muy importante que nuestra aplicación tenga en cuenta las directrices de una arquitectura REST.

Por otra parte, surge la necesidad de ofrecer desde el *backend* una API REST que devuelva información sobre los recursos que almacena en formato JSON. Esto se debe al carácter asíncrono de algunas pantallas, para lo cual el navegador tendrá que obtener datos de recursos (e.g., pases o jobs) para poder mostrarlos en la misma pantalla. Hemos intentado seguir las recomendaciones sobre APIs REST encontradas en fuentes como [15].

Resulta interesante, además, el hecho de que esta API pueda ser extendida en un futuro y utilizada por otras aplicaciones.

8.2 Patrones de diseño

8.2.1 Singleton

A veces es necesario tener sólo una instancia de un objeto, ya sea porque éste representa una entidad inherentemente única (e.g., un sistema de ficheros) o porque necesitamos un punto de acceso global a recursos internos o externos. Este último es nuestro caso: la aplicación precisa de unos parámetros globales, tales como la versión de *software* del sistema o el valor por defecto para sustituir cuando esperamos un `String` y la base de datos nos devuelve un `null`, que deben ser comunes para todos los módulos.

Esta necesidad se cubre gracias al patrón *singleton*, que garantiza la instanciación única de una clase. A continuación podemos ver un diagrama del diseño genérico de un *singleton*, junto con una implementación específica para esta aplicación:

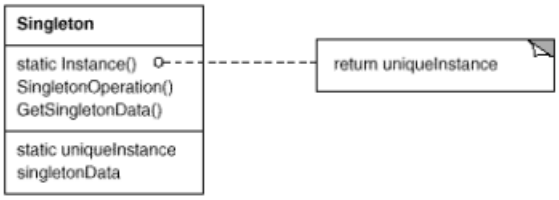

Figura 8.2: Representación genérica de un *singleton*.

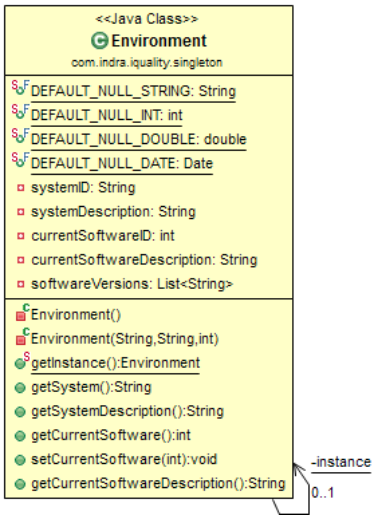

Figura 8.3: Representación del *singleton* `Environment`.

`Environment` guarda el identificador y descripción del sistema, identificador y descripción de la versión actual del *software*, y las otras versiones de software disponibles. Todos estos parámetros serán globalmente accesibles para cualquier módulo de la aplicación. Además, esta implementación hace homogénea la representación de valores nulos en la base de datos gracias a las constantes estáticas que define.

8.2.2 Front controller o Fachada

Estructurar diferentes módulos en un subsistema ayuda a reducir la complejidad. De hecho, muchas aplicaciones no desean acceso directo a los módulos específicos, sino a un subsistema abstracto que resuelva una tarea por ellas. Esto se puede conseguir mediante el patrón **Front controller**: introducimos un objeto que hace de **fachada** del subsistema, y es el único con el que otros sistemas pueden interactuar. La fachada será la encargada de delegar a los módulos internos.

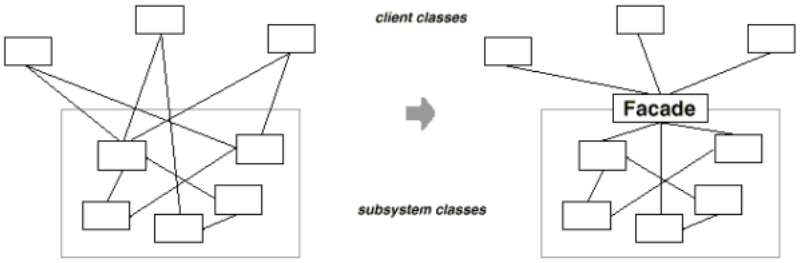

Figura 8.4: Representación genérica del patrón fachada.

Afortunadamente, este patrón es muy corriente en la mayoría de *framework*s de aplicaciones web.

8.2.3 Data Access Object/Data mapper

El acceso a los datos depende fuertemente de su fuente. Ya sea un modelo relacional, no relacional o un simple fichero de texto, el acceso a los datos no será el mismo. Ahora bien, queremos desacoplar la lógica de la aplicación de la capa de datos, de forma que seamos capaces de usar la misma lógica en diferentes entornos con otros métodos de persistencia. Este inconveniente lo soluciona el patrón DAO (Data Access Object), encapsulando y abstrayendo todo el acceso a la fuente de datos.

En nuestro caso, el papel del `BussinessObject` lo juega un controlador, que instancia un `DAO`. Éste accede a la fuente de datos (nuestra base de datos Oracle), y crea un objeto del modelo, por ejemplo un objeto `Job`. Esta práctica de asociar o *mapear* un *record* de la base de datos con un objeto en memoria es lo que se denomina **Data mapper**. Hemos tomado aún otra decisión de diseño: los DAOs son interfaces (por lo tanto, no pueden ser instanciadas y **deben** ser implementadas para usarlas), y el acceso se hace instanciándolos en otras clases. De esta manera, si cambiamos la fuente de datos, no habremos de cambiar los contratos de las operaciones ni la lógica de negocio en absoluto. Simplemente deberemos implementar las interficies de los DAOs para la nueva fuente de datos, y la aplicación funcionará sin siquiera notar el cambio.

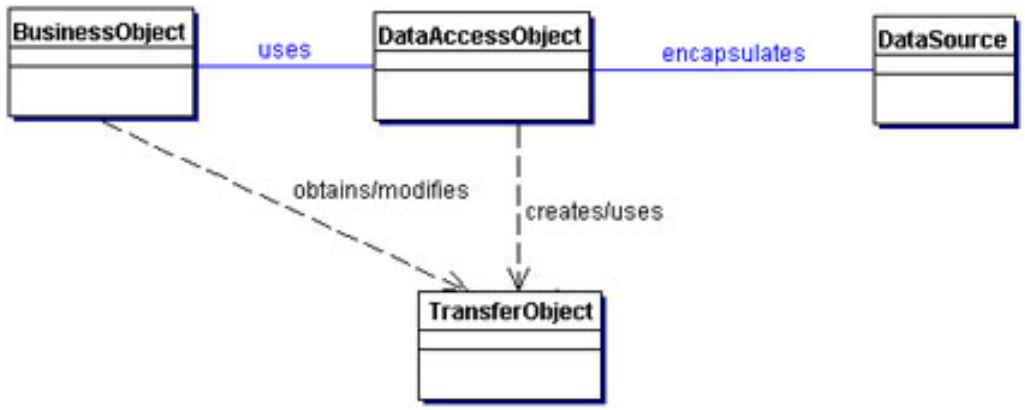

Figura 8.5: Representación del patrón DAO y Data mapper.

8.3 Tecnologías utilizadas

Ahora que conocemos los patrones y arquitectura que consideramos adecuados para aplicación, podemos decidir cuáles son las tecnologías que mejor se adaptan para desarrollar el proyecto, tanto en el *frontend* como en el *backend*.

8.3.1 Frontend

Bootstrap 3.3.2

Será una parte fundamental de la *vista* en el patrón MVC. Es un *framework* muy popular y utilizado, dado que viene equipado con multitud de clases CSS que facilitan muchísimo la implementación de una interfaz web estética, moderna y *responsive*.

jQuery 1.10.2

Indispensable librería de JavaScript que mejora y extiende sus funcionalidades. Además de constituir las vista del patrón MVC junto con Bootstrap, permite hacer peticiones asíncronas AJAX a la API Rest que deseamos implementar de forma muy sencilla. Asimismo, existe una gran variedad de *plug-ins* que se construyen sobre esta librería y ofrecen funcionalidades específicas y muy útiles para crear una interfaz dinámica y atractiva. Los *plug-ins* más importantes usados en el desarrollo son:

- DataTables 1.9.4: *plug-in* muy útil para manipular tablas; ofrece funcionalidades como paginación, búsqueda difusa e internacionalización.
- js-cookie 2.0.4: para crear y leer *cookies*.
- jsTree 3.2.1: indispensable para crear vistas en forma de árbol de directorios, altamente personalizable y con búsqueda difusa.
- jQuery validation 1.11.1: para validar formularios e informar de campos incorrectos.
- Select2 4.0.1: un selector de opciones personalizable.
- jQuery wizard 1.0: para mostrar formularios al estilo *wizard*.
- jQuery file upload 5.40.0: para cargar ficheros Excel al servidor.
- jQuery easy pie chart 2.1.3, jQuery sparkline chart 2.1.2, jQuery float chart 0.8.1: para generar gráficos de diferentes tipos.

8.3.2 Backend

Java 1.8 integrado en Spring 4.1.6

Dados los patrones vistos en la Sección 8.2, necesitamos un lenguaje de programación orientado a objetos para implementarlos. Partiendo de esa restricción, se ha elegido Java dado que es un lenguaje con el que estamos familiarizados, está extensamente documentado, es fácil encontrar ayuda o soluciones a problemas que puedan surgir durante el desarrollo y está integrado en varios *framework*s. De entre los muchos existentes, nos hemos decantado por Spring, uno de los más utilizados por la comunidad de desarrolladores para implementar aplicaciones web. Son muchas y variadas las razones por las que se ha elegido Spring.

Para empezar, el patrón MVC está asentado en el núcleo de Spring y hace que su utilización sea muy fácil:

- En la vista, podemos usar clases CSS y anotaciones que *mapean* elementos como campos de texto o selectores a parámetros pasados al controlador.
- Para que una clase de Java se comporte como un controlador, no hay más que adornarla con una anotación `@Controller`. Para indicar las rutas que tratará, usamos la anotación `@RequestMapping`, y para tomar los parámetros pasados en la URL siguiendo el formato REST (ver Subsección 8.1.2) simplemente usamos la anotación `@PathVariable`.

- En cuanto a la persistencia de datos, Spring ofrece librerías que abstraen sobre la popular JDBC, de manera que el desarrollador no se tiene que preocupar de manejar conexiones con la base de datos y se puede concentrar en la lógica de la aplicación.

Por ejemplo, para declarar un controlador hacemos lo siguiente:

```
35 @Controller
36 @RequestMapping("/pases")
37 public class ExecutionManagementController {
```

En la línea 35 decimos que la clase `public class ExecutionManagementController` será un controlador, y en la línea 36 especificamos que se encargará de todas las peticiones cuya ruta comience por `/pases`.

Más adelante, dentro de la misma clase, podemos definir un método:

```
143         @RequestMapping(value = "/{idEjecucion}/jobs", method =
            ↪  RequestMethod.GET)
144         private String getJobsOfExecution(@PathVariable int idEjecucion,
            ↪  ModelMap model) {
```

En la línea 143 indicamos que este método servirá peticiones `HTTP GET` a la ruta `/pases/<identificador>/jobs` (ya que está en la clase que sirve las peticiones que comienzan con `/pases`), y en la línea 144 tomamos el parámetro `idEjecucion` pasado en la URL para usarlo en el cuerpo del método.

Como segunda ventaja de Spring podemos destacar que implementa el patrón fachada de forma nativa. En su nomenclatura, el *front controller* es un `DispatcherServlet`, pero el papel que juega es el mismo:

El `DispatcherServlet` recibe todas las peticiones y las delega al controlador indicado. El controlador realiza la lógica pertinente y devuelve el control al `DispatcherServlet`, que genera la vista y devuelve una respuesta.

Finalmente, otra importante ventaja de Spring es que está magníficamente documentado [8], además de ofrecer una gran cantidad de guías y tutoriales [21], lo cual ha resultado esencial para el desarrollo[1].

El IDE utilizado es Spring Tool Suite 3.7.2, una versión específica de Eclipse para Spring construida sobre Eclipse Mars 4.5.1.

[1] No hemos aprovechado más que una minúscula parte del abanico de funcionalidades que ofrece Spring: módulos de seguridad, de *testeado*, para generar vistas, cacheado, vista de documentos, etc.

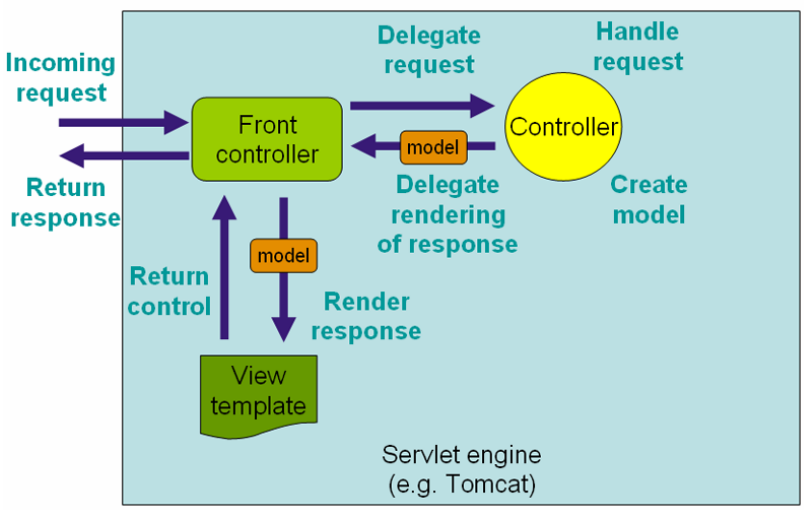

Figura 8.6: Representación del patrón fachada en Spring.

Oracle DB 11g

Es el entorno de datos que se usaba para la versión anterior de la aplicación, y no se cambiará por el momento. Accederemos a los datos del Data mart mediante la interfaz ofrecida por SQLDeveloper.

8.4 Capa de presentación

8.4.1 Diseño externo

Como ya expusimos en la Sección 3.1, uno de los propósitos del proyecto es mejorar el aspecto estético y la usabilidad de la aplicación. Para ilustrar este punto, obsérvese una pantalla de la versión antigua de *iQuality* en la Figura 8.7.

La interfaz, aunque plenamente funcional, no es moderna y resulta poco atractiva. Además la aglomeración de datos puede resultar un tanto ofuscadora: en general no nos interesa ver tres tablas en una misma pantalla. Las tecnologías descritas en la Subsección 8.3.1 nos permiten mejorar mucho el *look & feel* de la aplicación, como se puede ver en la Figura 8.8.

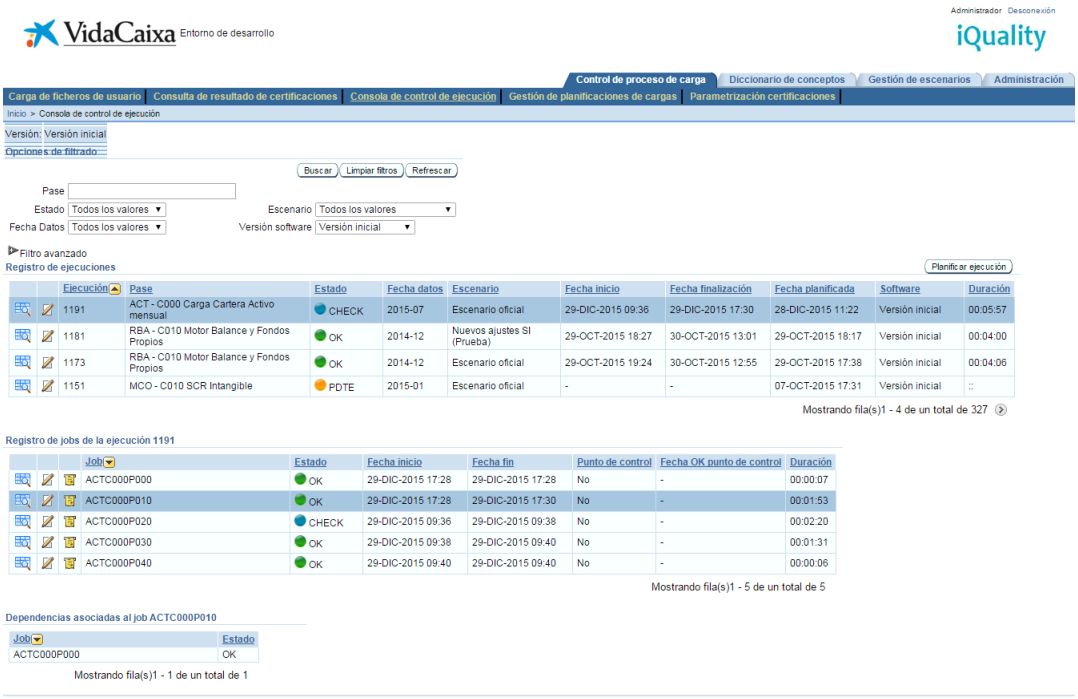

Figura 8.7: Pantalla de control de ejecuciones en la versión anterior de *iQuality*.

8.4.2 Mapa navegacional

Con la nueva apariencia de la aplicación, presentamos a continuación las pantallas por las que un usuario puede navegar.

Pantalla inicial

Como podemos ver en la , esta es la página principal de la aplicación. Aquí el usuario tiene una visión global de la calidad de los datos del sistema mediante diferentes gráficas y diagramas[2]. Por otra parte, desde el menú lateral (disponible en cualquier pantalla), el usuario puede ir a: diccionario de conceptos, reglas de calidad, control de procesos, administración, iniciar sesión o al menú de ayuda[3].

[2] En el momento de escribir esto, las gráficos y diagramas no usan aún la lógica de la aplicación, pero ofrecen una ilustración del tipo de resultado a esperar en futuras iteraciones.
[3] El menú de ayuda está aún por implementar.

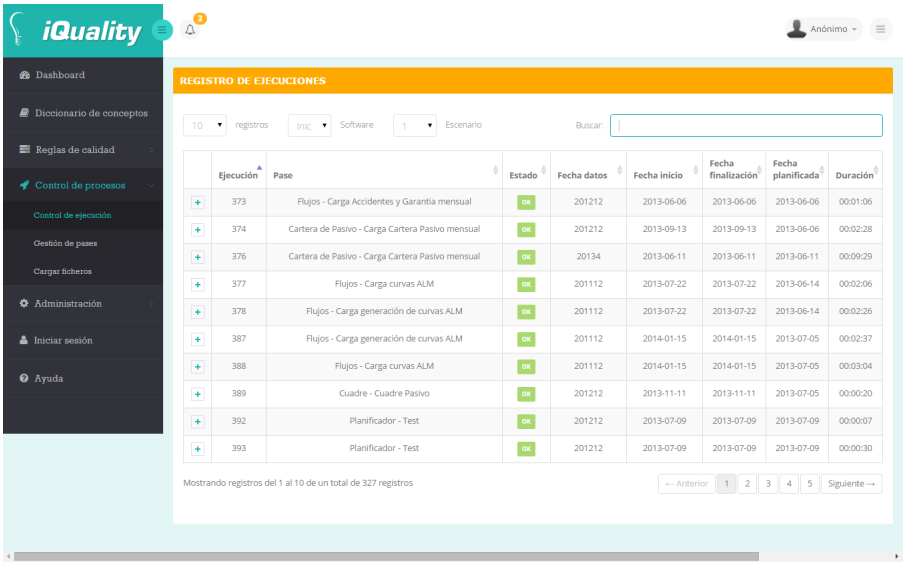

Figura 8.8: Pantalla de la nueva versión de *iQuality*, equivalente a la de la Figura 8.7.

Diccionario de conceptos

En esta pantalla el usuario puede navegar por el árbol de conceptos o usar el buscador para filtrar. Haciendo click en conceptos del tipo **indicador** o **atributo**, se obtienen los detalles del concepto en el panel derecho (Figura 8.10).

Si el concepto es un atributo, tendremos dos pestañas: una con datos básicos y otra con datos de entidad (Figura 8.11).

Además, si el atributo tiene una tabla maestra, podremos elegir entre ver los datos de entidad del concepto o de su maestro (Figura 8.12).

Si el concepto es un indicador, además de las pestañas de datos básicos y de entidad, podremos consultar una pestaña con las reglas de certificación que aplican a este indicador (Figura 8.13).

Reglas de calidad

En cuanto a las reglas de calidad, podemos hacer dos cosas: consultarlas o parametrizarlas. Esto lo podemos ver en la Figura 8.14 y la Figura 8.15, respectivamente. Nótese que en el segundo caso, un formulario de tipo *wizard* se encarga de controlar y guiar al usuario por el proceso de creación de una nueva regla.

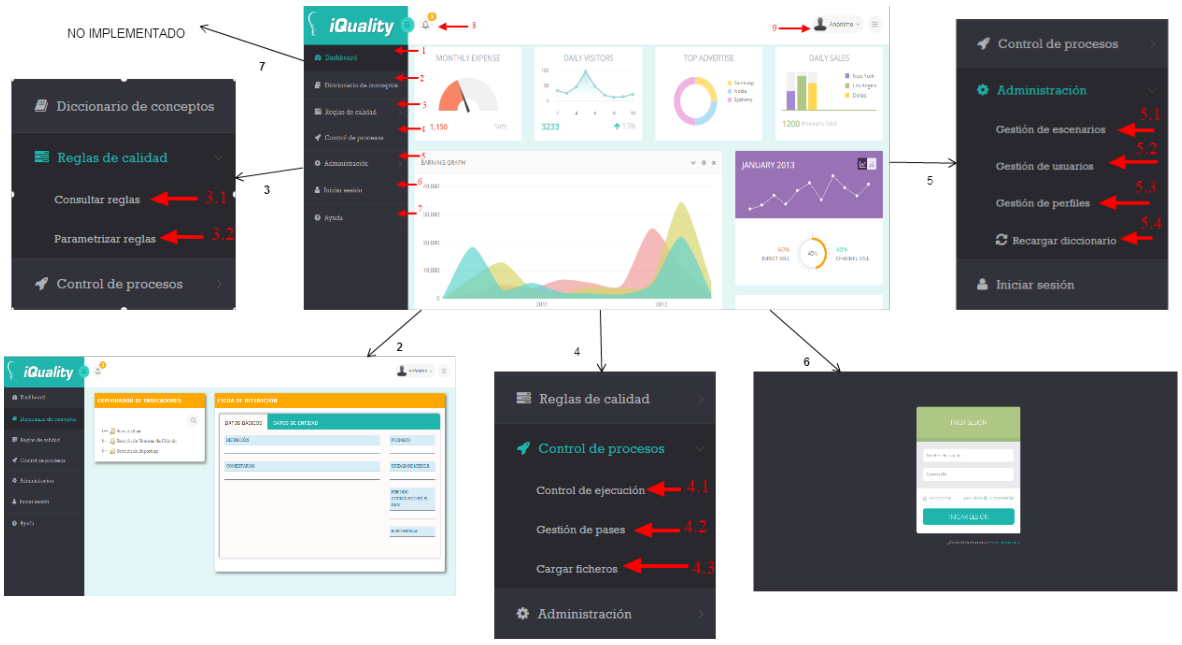

Figura 8.9: Diagrama de navegabilidad desde la página inicial.

Control de procesos

Desde este submenú tenemos acceso a las pantallas de control de ejecución (Figura 8.16), gestión de pases (Figura 8.17) y carga de ficheros (Figura 8.18).

Administración

Desde este submenú de administración (Figura 8.19) podemos gestionar escenarios, perfiles, usuarios, y recargar el diccionario de conceptos[4], aunque por el momento la única de estas funcionalidades que ha sido implementada es la última.

Iniciar sesión

Desde esta pantalla (Figura 8.20) el usuario puede identificarse en la aplicación. Esto no era una prioridad del proyecto, de modo que se ha implementado la interfaz pero no la funcionalidad; esa será una tarea para futuras iteraciones.

[4]Para entender por qué es necesario refrescar el diccionario, véase la Subsección 9.3.3.

Figura 8.10: Diagrama de navegabilidad desde la pantalla del diccionario de conceptos.

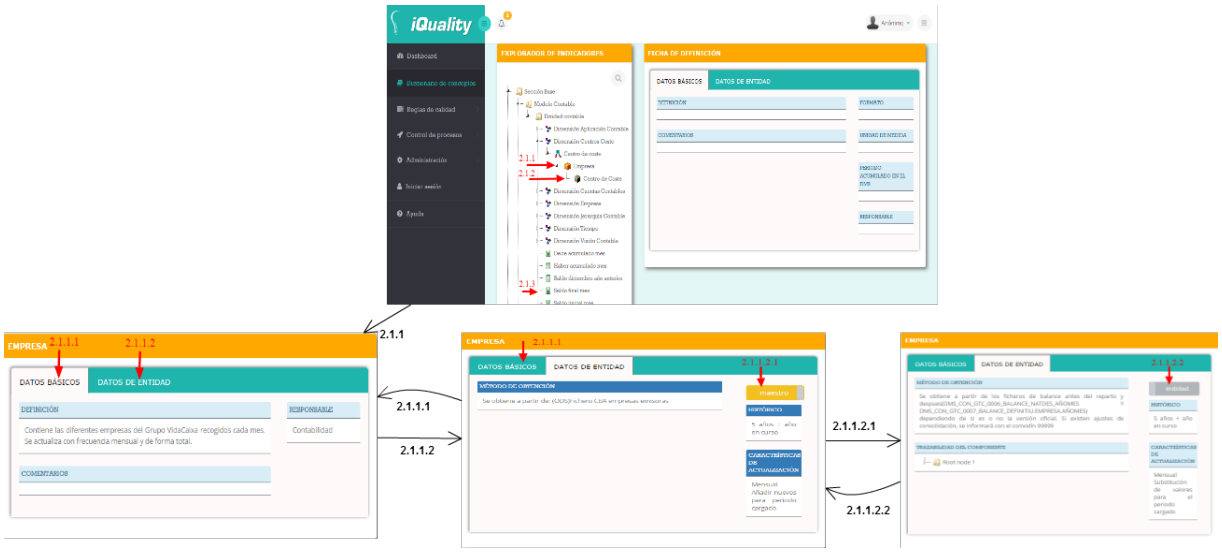

Figura 8.11: Diagrama de navegabilidad para la vista de un concepto del tipo atributo con tabla maestra.

Figura 8.12: Diagrama de navegabilidad para la vista de un concepto del tipo atributo sin tabla maestra.

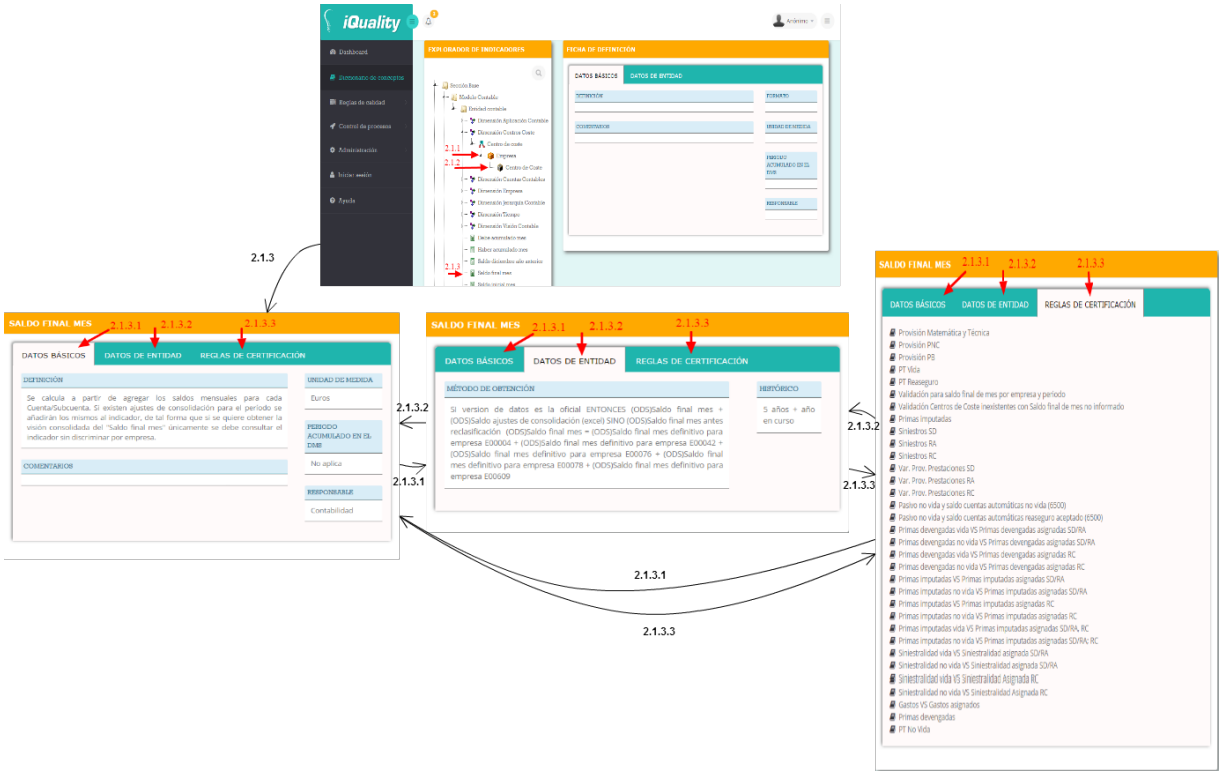

Figura 8.13: Diagrama de navegabilidad para la vista de un concepto del tipo indicador.

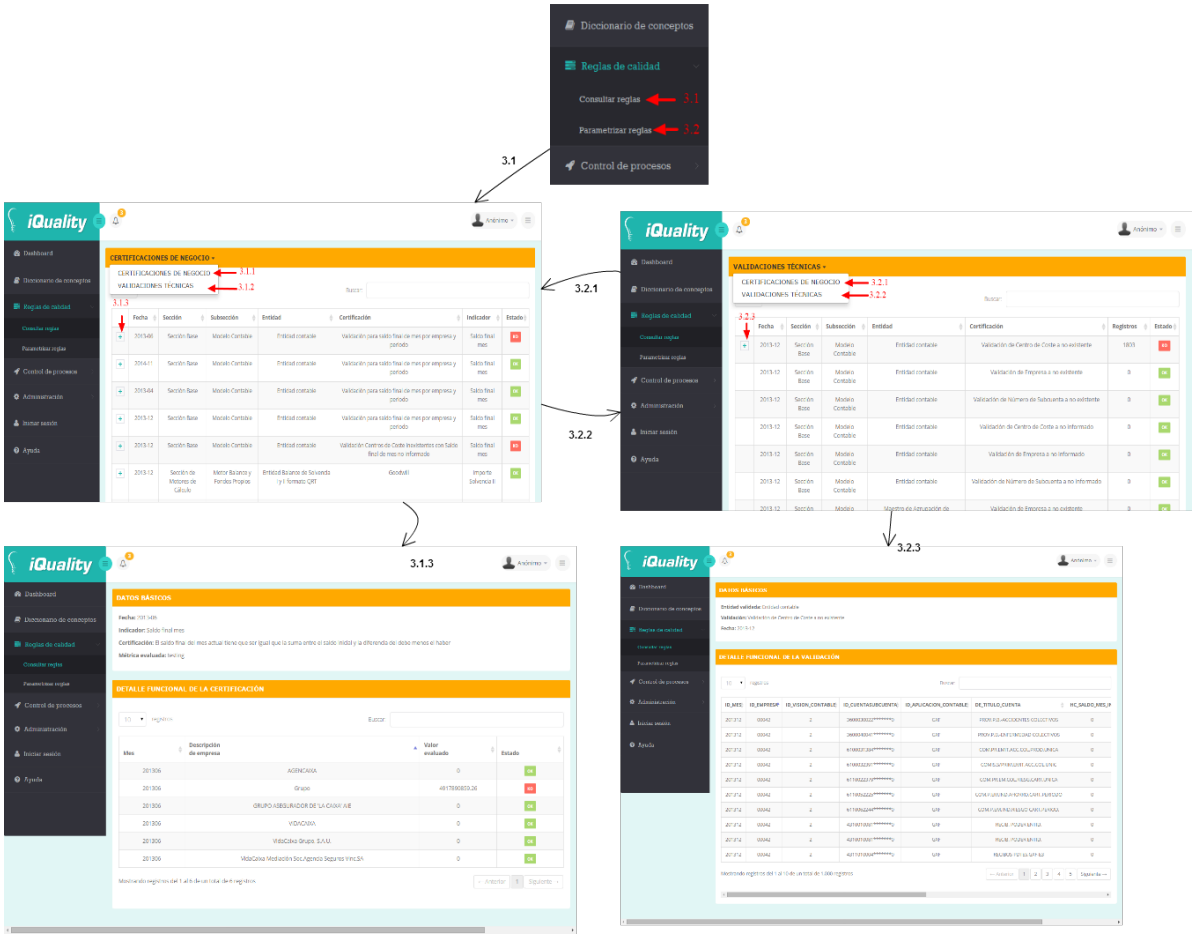

Figura 8.14: Diagrama de navegabilidad desde la pantalla de consulta de reglas.

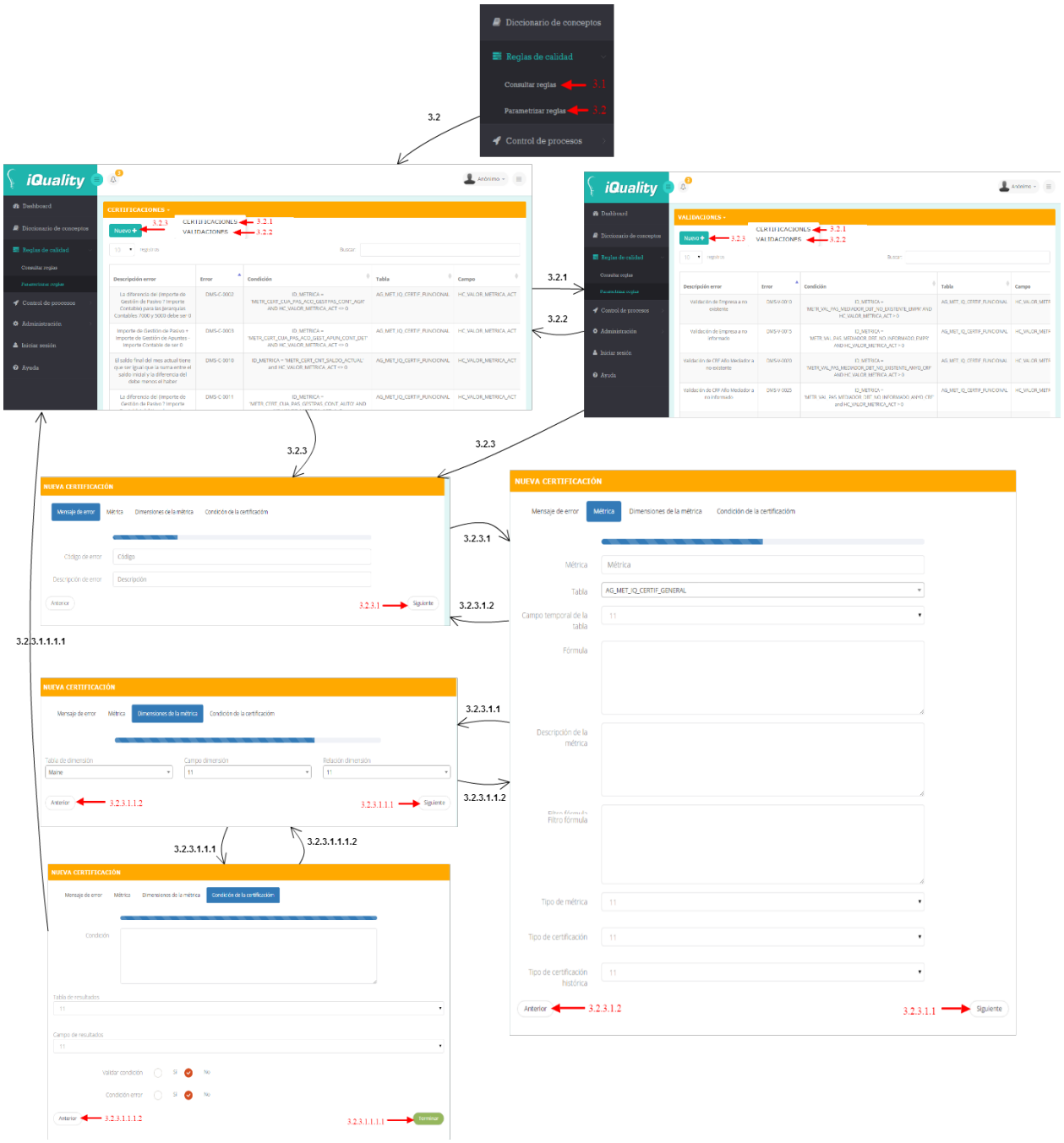

Figura 8.15: Diagrama de navegabilidad desde la pantalla de parametrización de reglas.

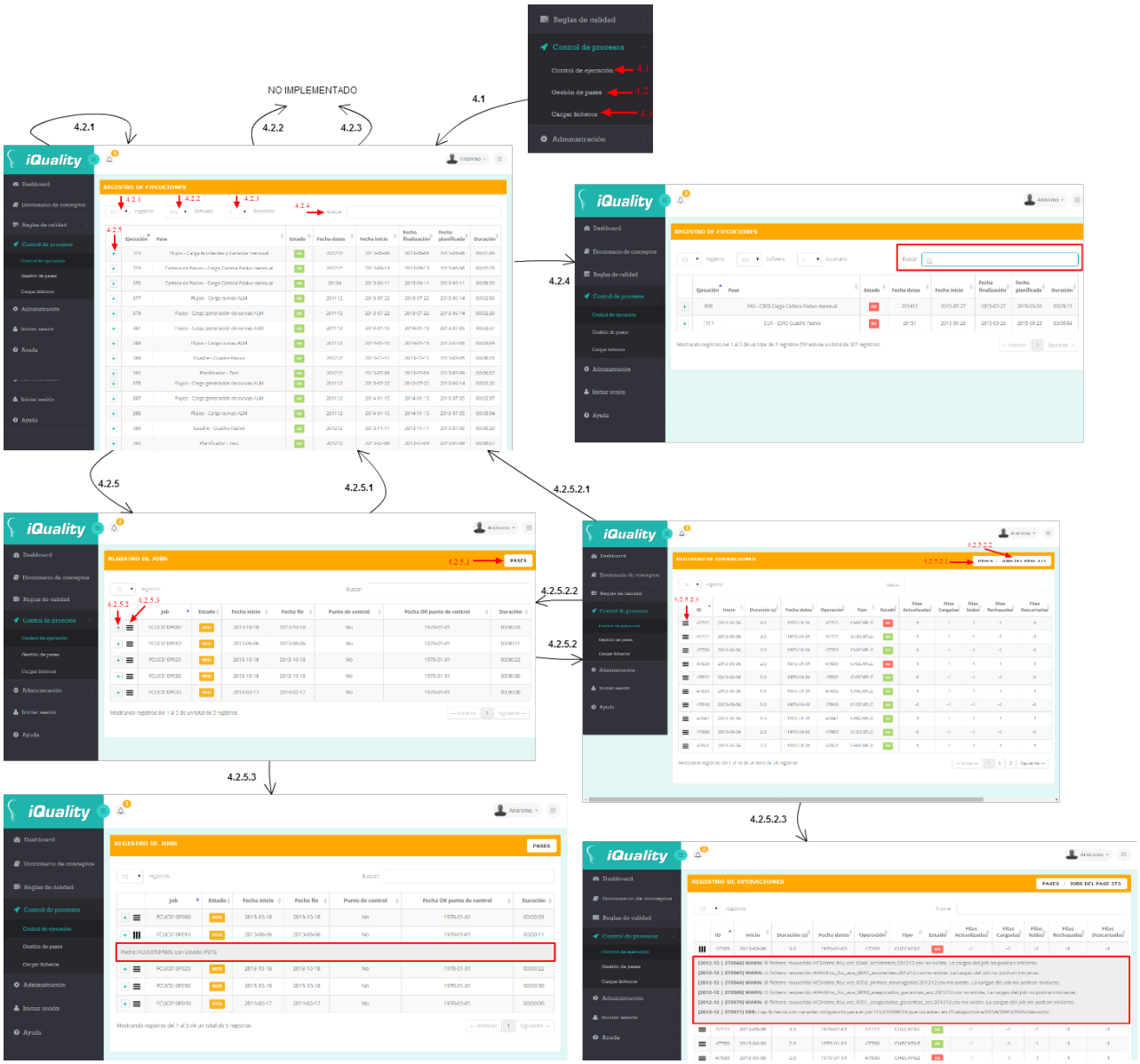

Figura 8.16: Diagrama de navegabilidad desde la pantalla de control de ejecución.

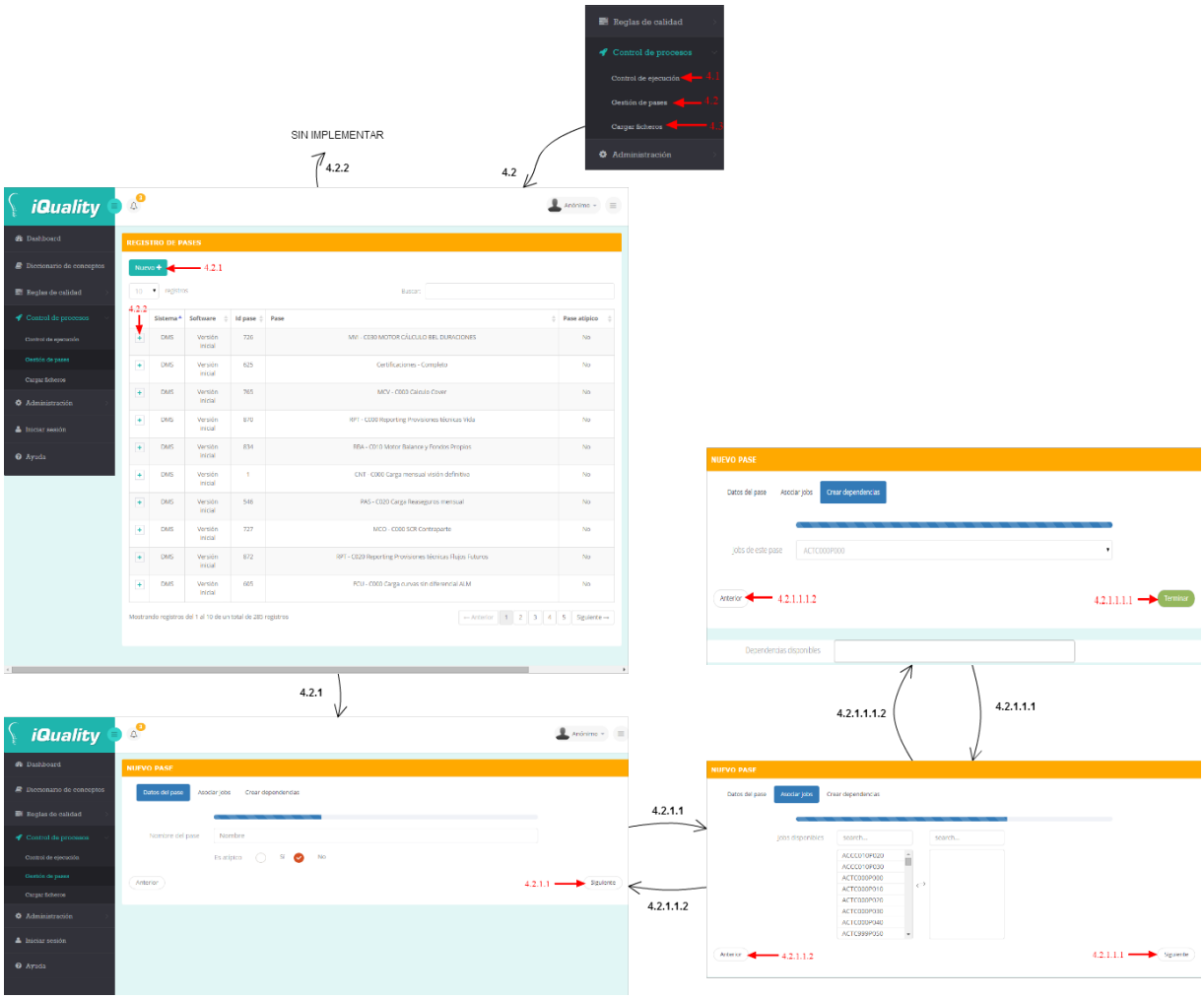

Figura 8.17: Diagrama de navegabilidad desde la pantalla de gestión de pases.

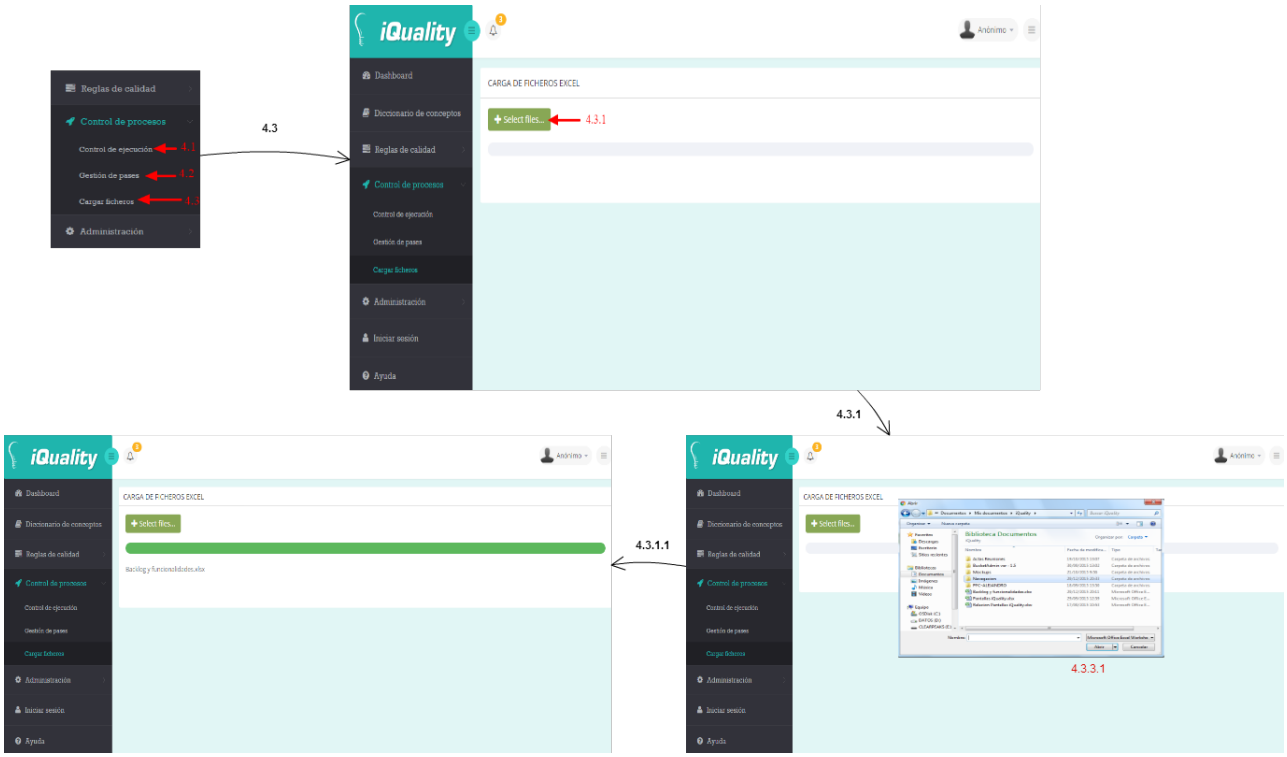

Figura 8.18: Diagrama de navegabilidad desde la pantalla de carga de ficheros.

Figura 8.19: Diagrama de navegabilidad desde la pantalla de administración.

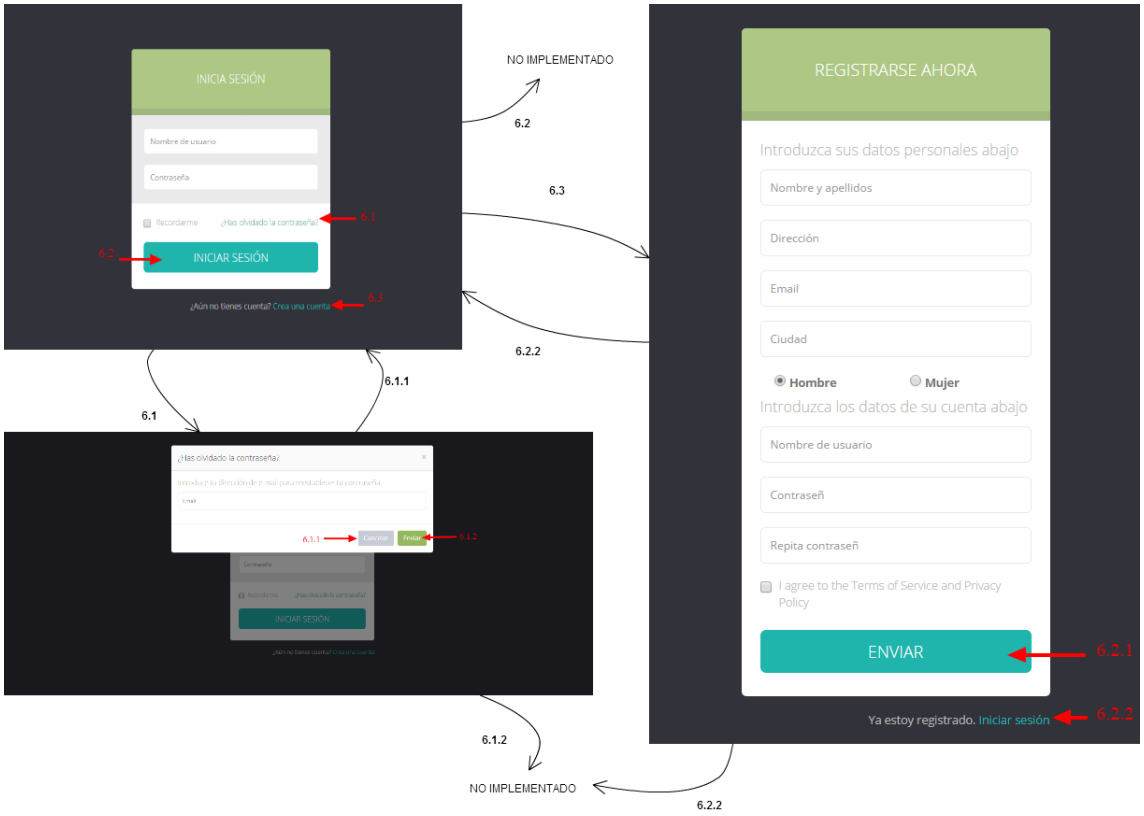

Figura 8.20: Diagrama de navegabilidad de la pantalla de inicio de sesión.

8.5 Capa de dominio

8.5.1 Modelo conceptual

Estas son las clases del modelo, es decir, las del `package com.indra.iquality.model` (véase la Sección 9.1)[5]. Como podemos ver en la Figura 8.21, tenemos tres tipos de entidad importantes:

- Conceptos del diccionario: tienen datos necesarios para su gestión, y una descripción, que puede ser diferente para un indicador, atributo o atributo con tabla maestra.
- Certificados: pueden ser de negocio o técnicos, y pueden contener más información en forma de detalles.
- Ejecuciones: tienen algún pase de definición asociado, el cual está compuesto de *jobs*. Éstos, a su vez, pueden tener dependencias entre sí y un registro de operaciones, para poder seguirles la traza.

Los controladores, por otro lado, no forman parte del modelo, pero sí de la lógica: interactúan con la vista y el modelo. Por eso es interesante tener una noción de ellos, presentada en la Figura 8.22. La descripción del comportamiento de los controladores se verá en la Subsección 8.5.2.

[5]Para una vista ampliada de este diagrama, véase el Apéndice A

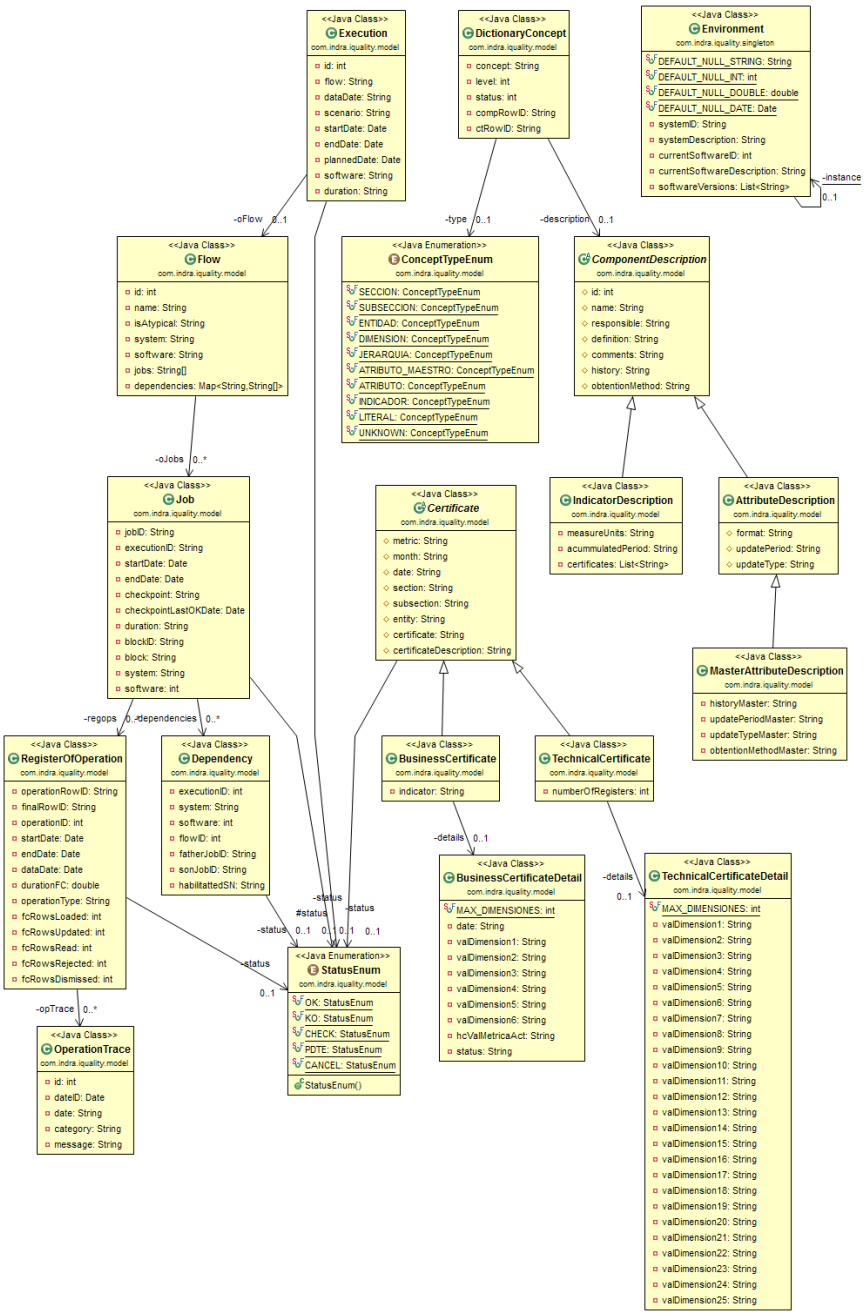

Figura 8.21: Diagrama UML de las entidades del modelo.

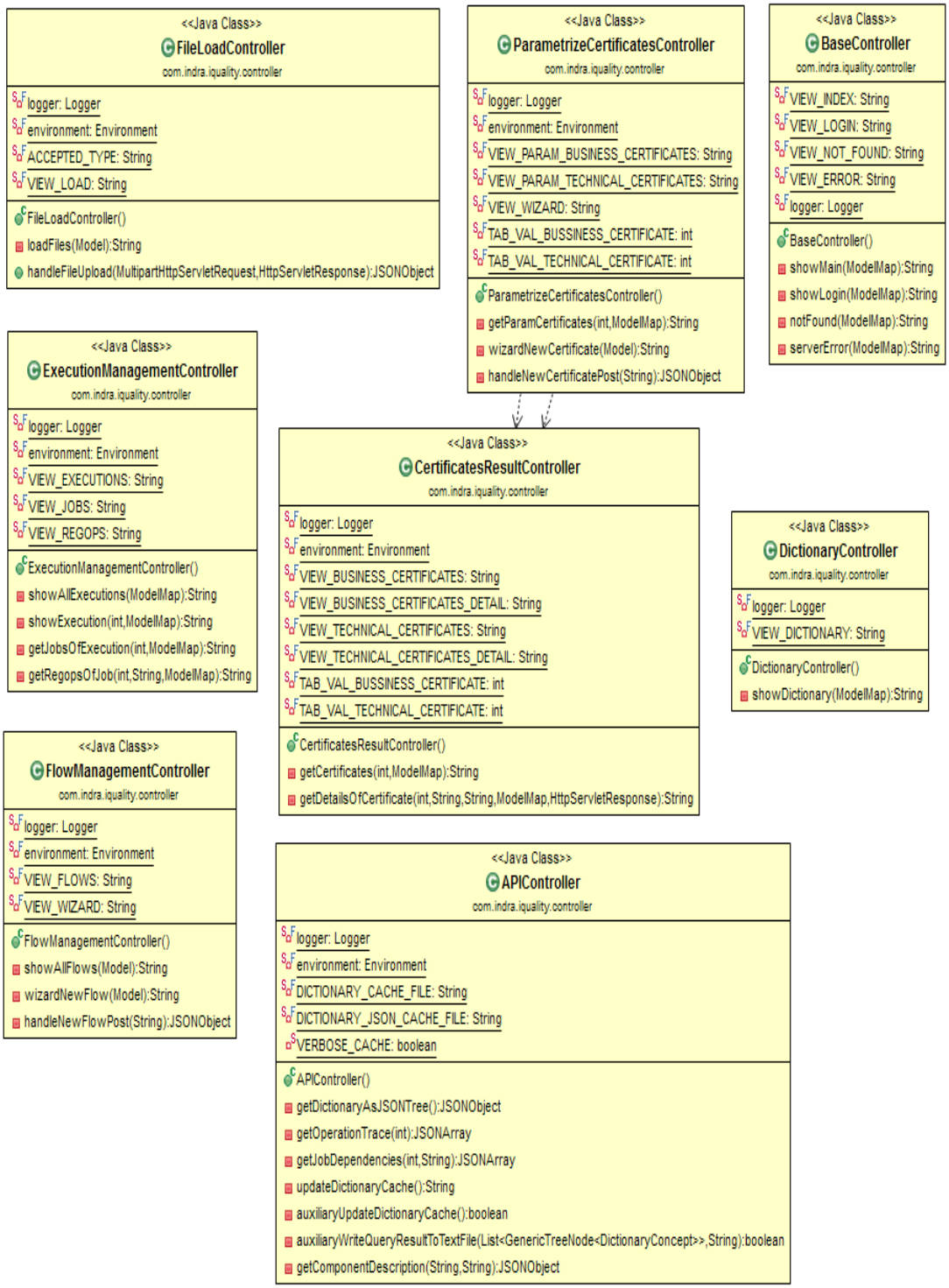

Figura 8.22: Diagrama UML de los controladores.

8.5.2 Modelo de comportamiento

Al tratarse de una aplicación web, el comportamiento del sistema sigue siempre el mismo patrón: el usuario selecciona alguna opción, el controlador fachada redirige la petición al controlador responsable, éste crea los modelos necesarios y hace las operaciones, genera una vista con el resultado y devuelve el control al usuario.

Se había considerado en un principio representar estos comportamientos mediante los clásicos diagramas de secuencia, pero los juzgamos demasiado complejos y excesivamente informativos. Lo que nos interesa es saber a alto nivel, pero sin perder rigurosidad, cómo se comunican los diferentes módulos para llevar a cabo una historia de usuario. Por todo esto, decidimos finalmente usar diagramas BPMN: son más fáciles de entender, proporcionan toda la información necesaria y se integran mejor con el desarrollo ágil seguido.

Los diagramas presentados a continuación corresponden a las siguientes historias de usuario (véase la Sección 7.1): ver todas las ejecuciones (historia de usuario #1), ver los jobs de una ejecución y sus detalles (historias de usuario #2 y #3), ver todos los pases (historia de usuario #4), crear un nuevo pase (historia de usuario #5), ver todas las certificaciones y sus detalles (historias de usuario #6, #7, #8 y #9), crear una nueva certificación (historia de usuario #10), cargar ficheros Excel (historia de usuario #11), consultar el diccionario de conceptos, con actualizar caché y cargar árbol como subprocesos (historias de usuario #12 y #13), y finalmente recibir alertas de ejecuciones fallidas (historia de usuario #14).

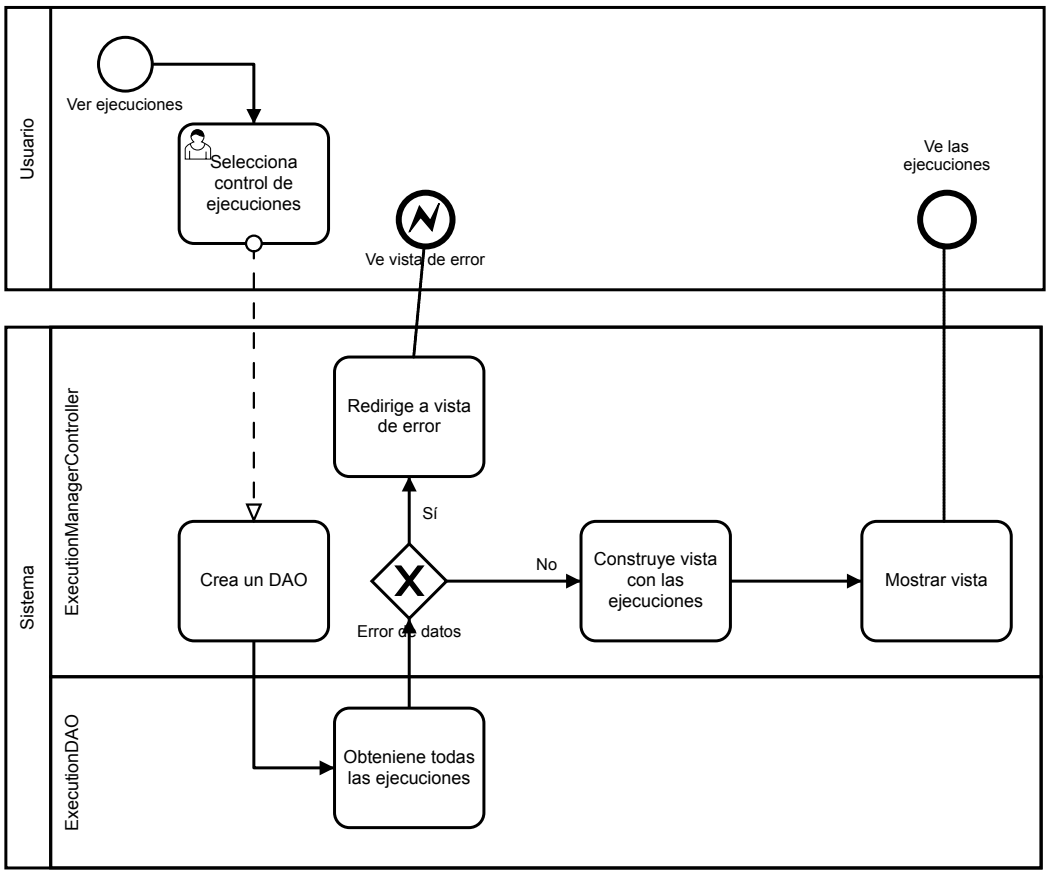

Figura 8.23: Diagrama BPMN para ver las ejecuciones.

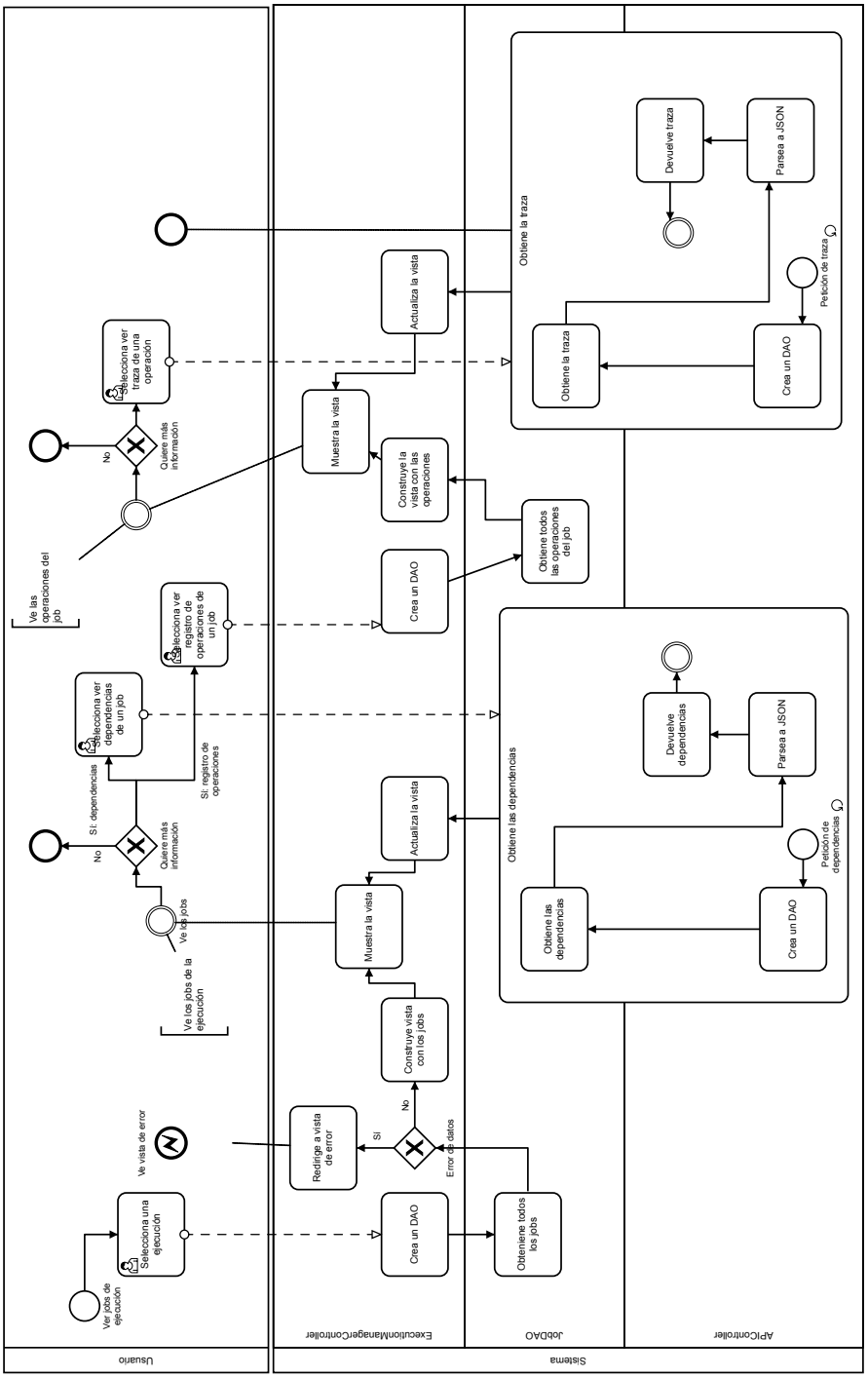

Figura 8.24: Diagrama BPMN para ver los jobs de una ejecución y sus detalles.

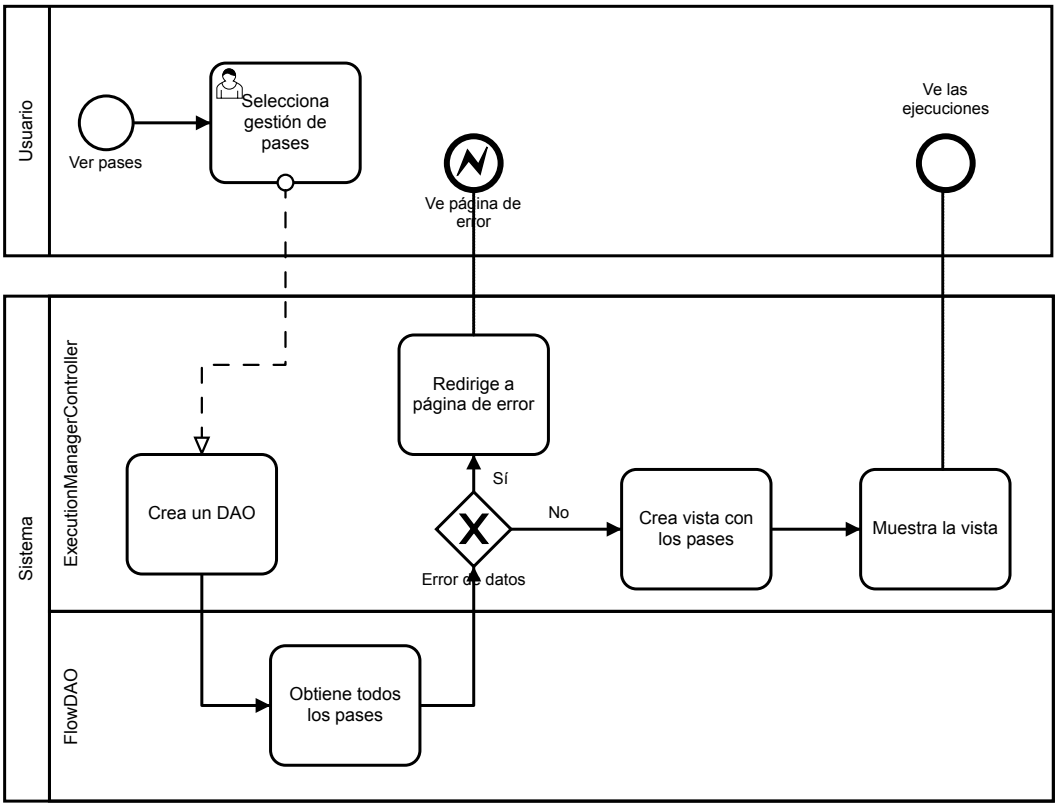

Figura 8.25: Diagrama BPMN para ver los pases.

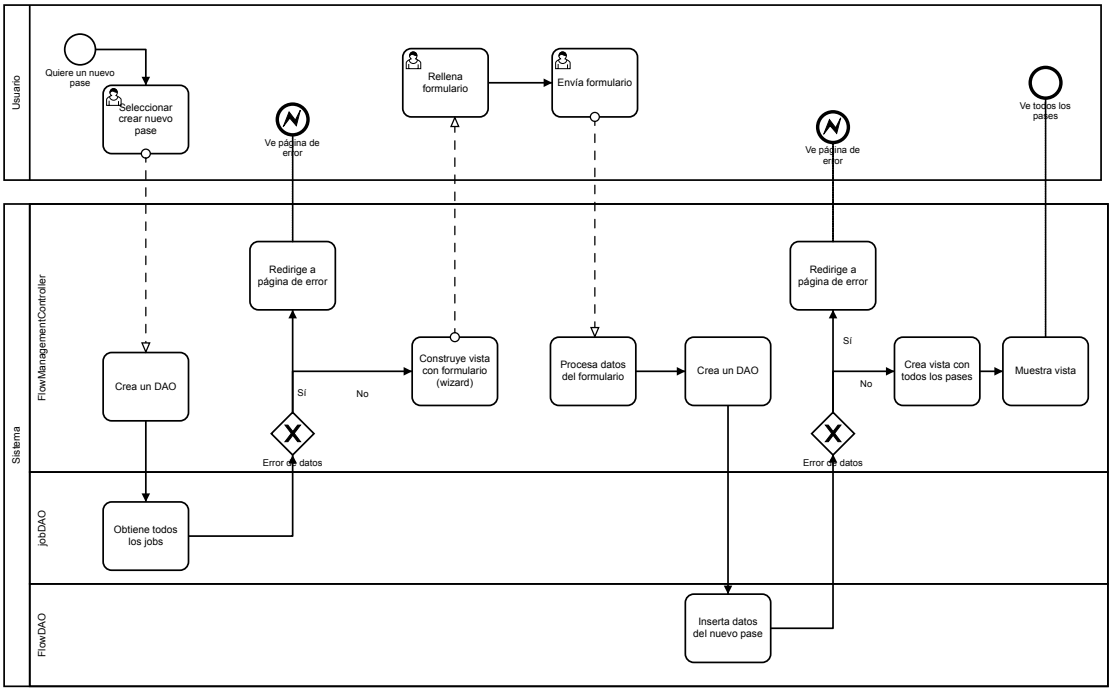

Figura 8.26: Diagrama BPMN para crear un nuevo pase.

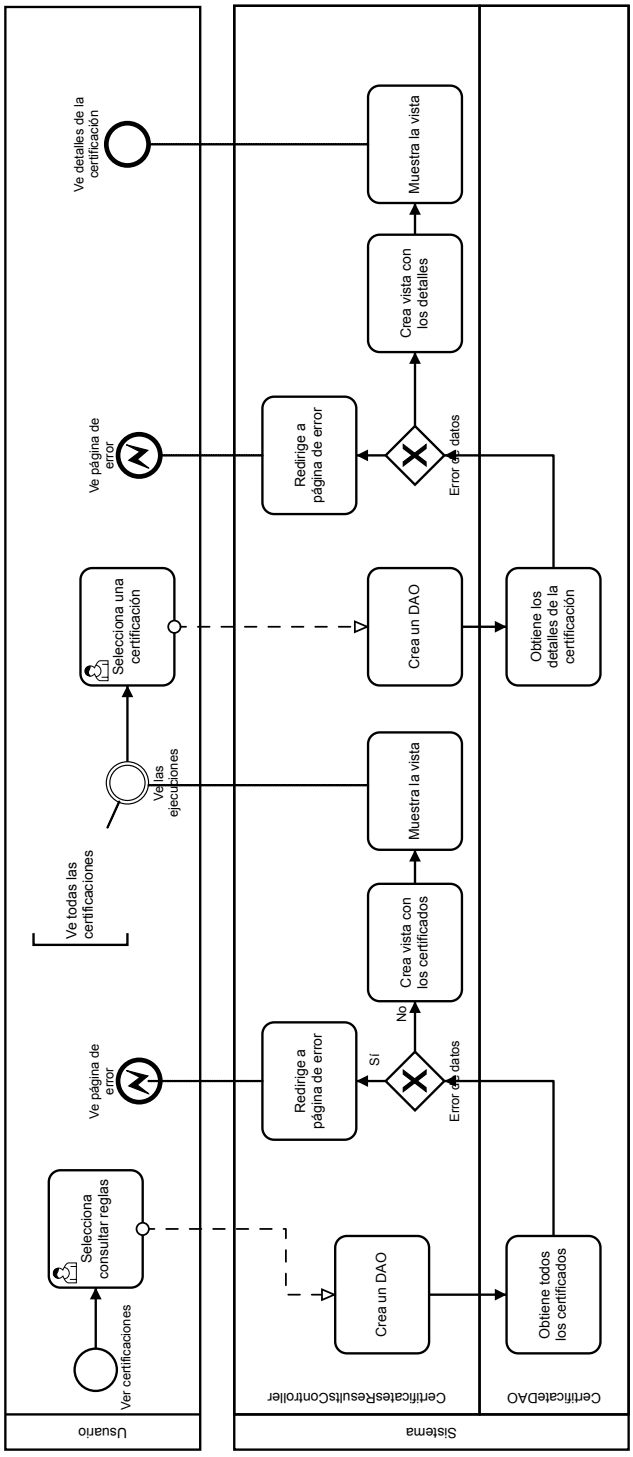

Figura 8.27: Diagrama BPMN para ver las certificaciones.

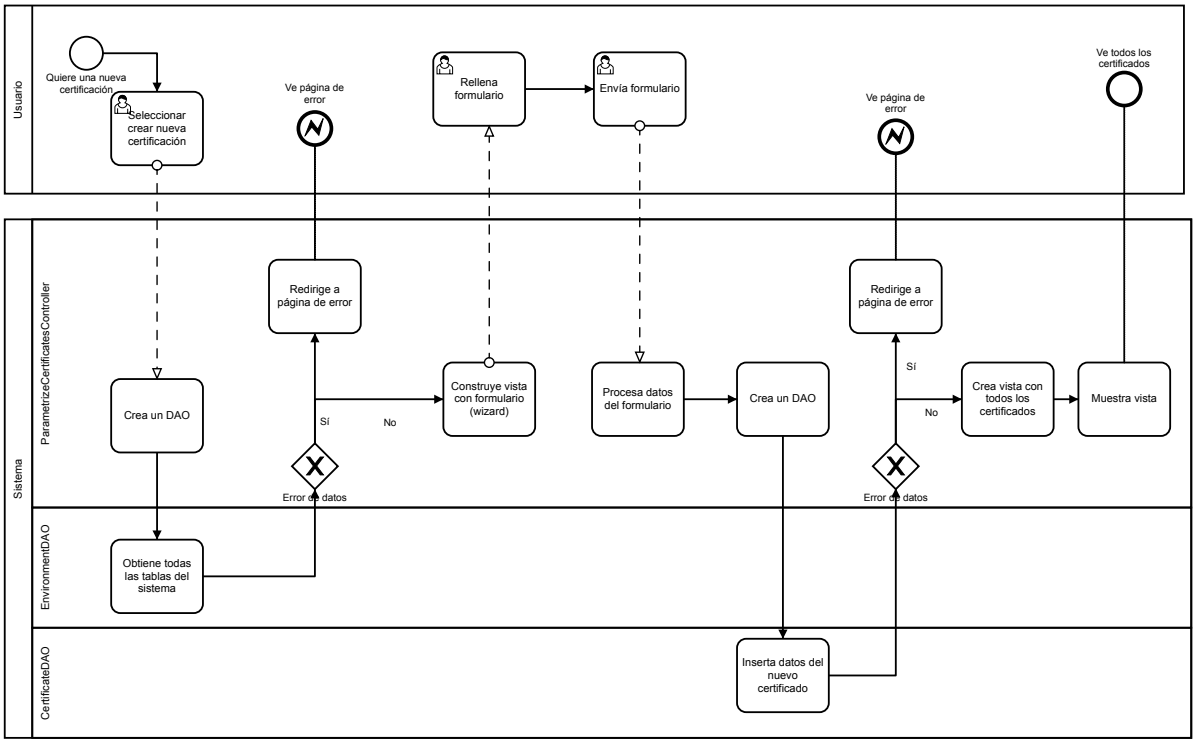

Figura 8.28: Diagrama BPMN para crear una nueva certificación.

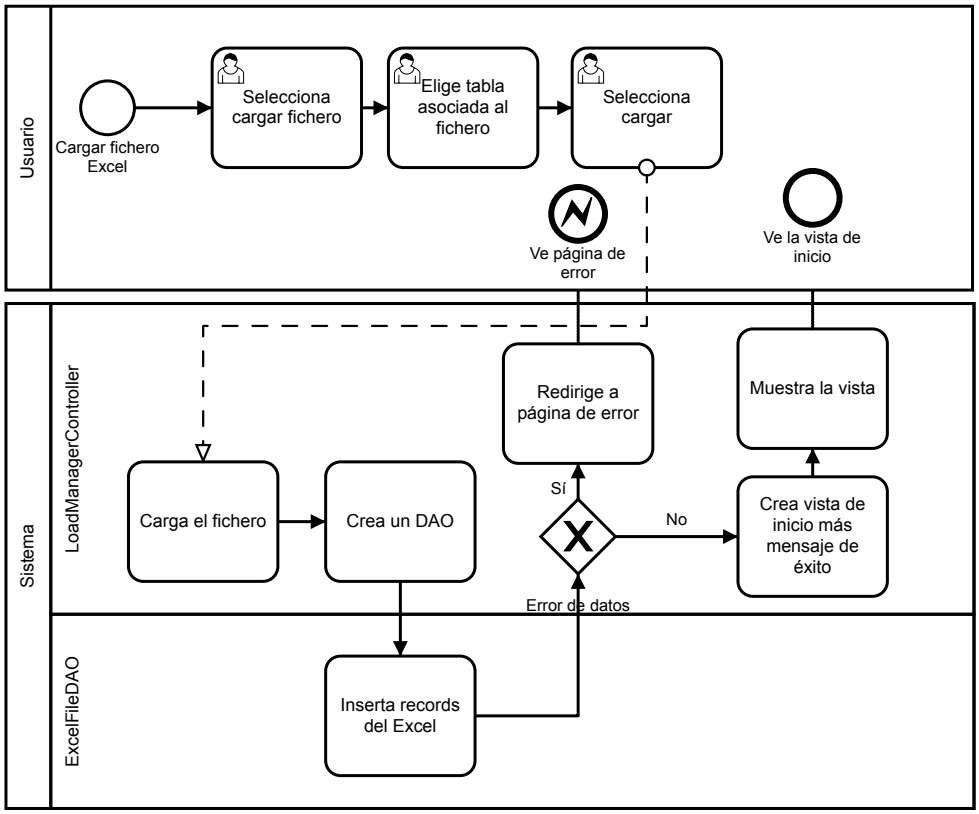

Figura 8.29: Diagrama BPMN para cargar ficheros Excel.

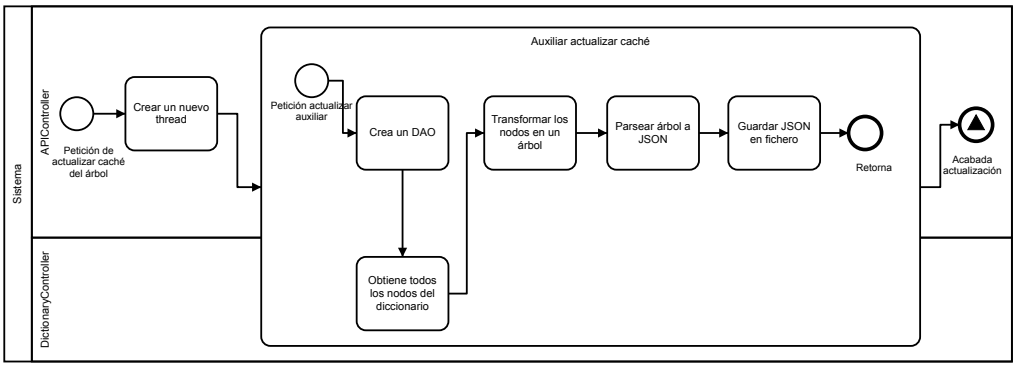

Figura 8.30: Diagrama BPMN para actualizar la caché del diccionario.

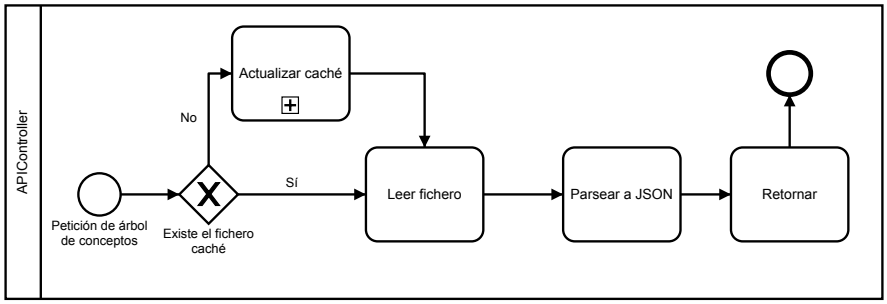

Figura 8.31: Diagrama BPMN para cargar el árbol del diccionario.

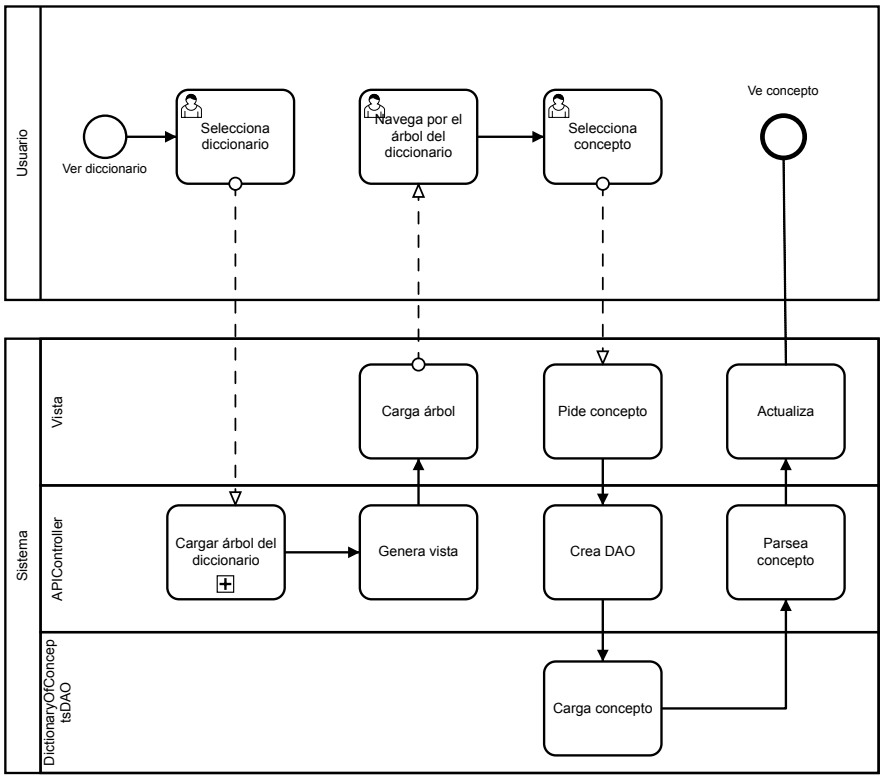

Figura 8.32: Diagrama BPMN para consultar conceptos del diccionario.

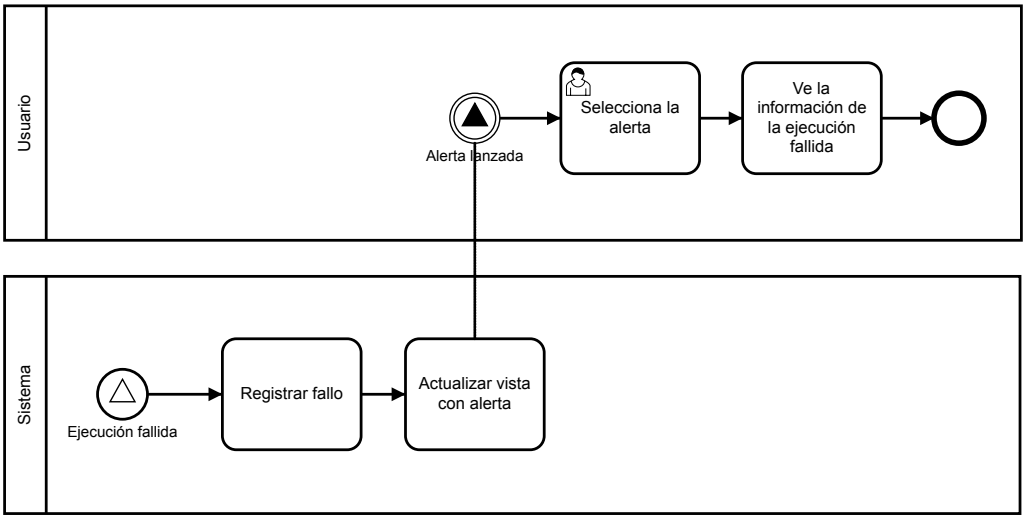

Figura 8.33: Diagrama BPMN para recibir alertas de ejecuciones fallidas.

8.6 Capa de datos

Como se recordará de la Sección 3.1, el refactorizado del Data mart queda fuera del alcance del proyecto debido a su complejidad. Por este mismo motivo, resulta inviable presentar una lista de los **cientos** de tablas del Data mart, y poco informativo describir las que se han utilizado. En consecuencia, nos limitamos a presentar y describir en la Figura 8.34 el diagrama UML de los *data mappers*[6].

Como ya habíamos visto en la Subsección 8.2.3, vemos que los DAOs son interfaces implementadas por clases concretas para el caso específico de nuestra base de datos. Además, dado que las implementaciones tenían algunos atributos y métodos en común (especialmente la fuente de datos), los hemos abstraído en una clase de la que heredan.

Tenemos DAOs para *mapear* certificaciones y sus detalles (`BusinessCertificateDAOJDBCTemplateImpl`), para obtener pases, ejecuciones y jobs(`FlowDAOJDBCTemplateImpl`, `ExecutionDAOJDBCTemplateImpl` y `JobDAOJDBCTemplateImpl`), para los conceptos del diccionario, etc.

[6]Para una vista ampliada de este diagrama, véase el Apéndice A

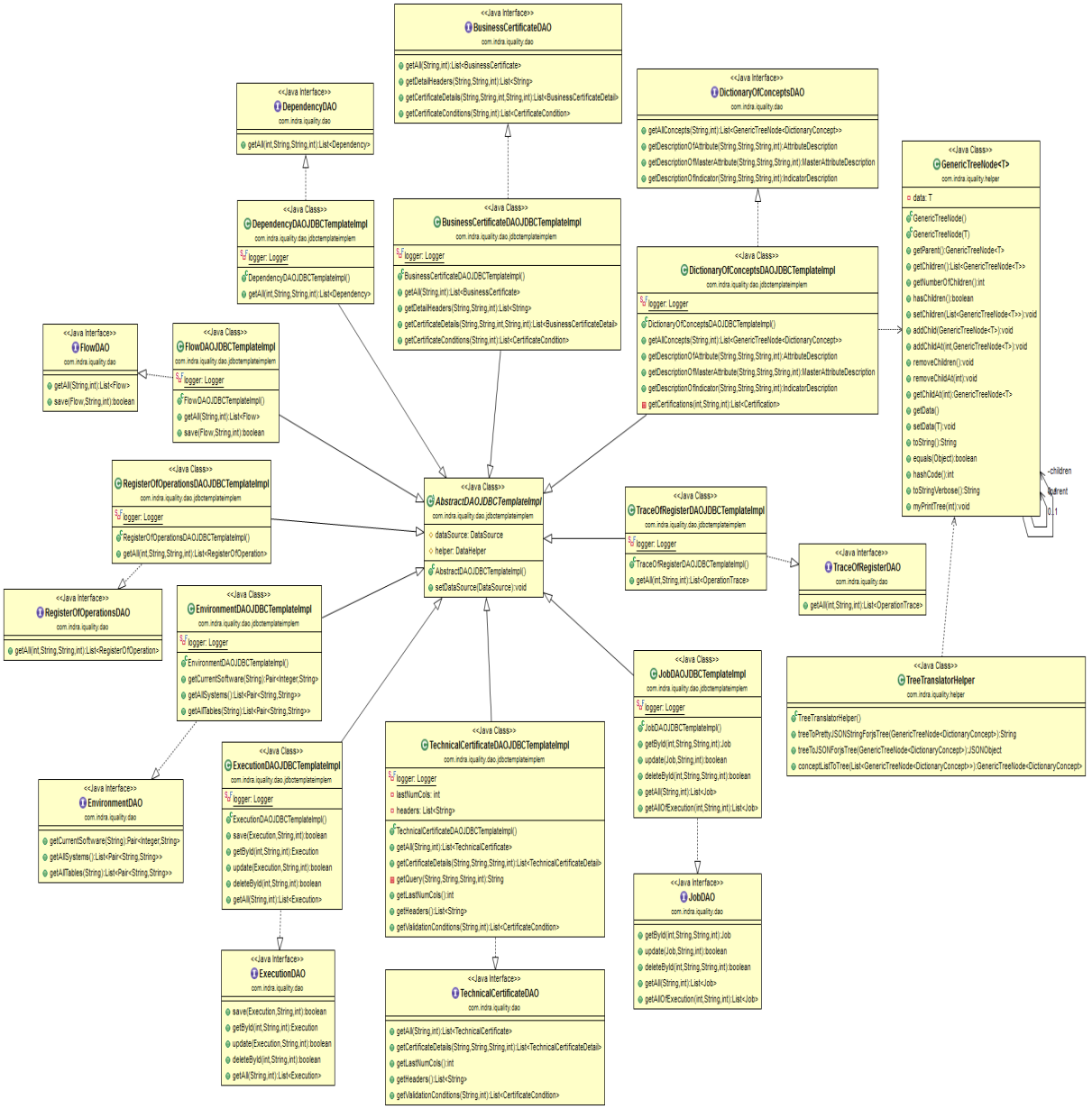

Figura 8.34: Diagrama UML de las interfaces de datos.

9 Implementación

Antes de hablar de la implementación, debemos tener en mente algunos aspectos del proyecto: entornos, organización estructural de las clases, y el carácter iterativo del desarrollo.

En cuanto a los entornos, debemos precisar que nos hemos limitado a dos: entorno de desarrollo y de preproducción. El entorno de desarrollo consta del ordenador portátil en el que hemos hecho todo el trabajo, así como los *tests*, copiando algunos datos del Data mart a una base de datos local de pruebas. El entorno de preproducción lo hemos utilizado regularmente (una vez por semana aproximadamente) para comprobar que la aplicación funciona con datos reales en un entorno muy similar al de producción. Cabe destacar que los entornos de integración y producción no los hemos utilizado en absoluto, el primero porque al ser un proyecto individual no era necesario integrar nuestros cambios con los de otros desarrolladores, y el segundo porque aún no es momento de sustituir la versión de *iQuality*.

Lo referente a la organización de las clases implementadas lo veremos en detalle en la siguiente sección, pero podemos resumir la estructura de la siguiente manera: las clases se han agrupado en paquetes relacionados semánticamente en cuanto a la lógica de negocio de la aplicación. Hay clases que representan objetos del modelo de negocio, clases para acceder a los datos, clases auxiliares, clases para controlar peticiones y flujo de datos, etc.

Finalmente, como ya habíamos introducido en la Sección 3.3 y descrito más extensamente en la Subsección 4.2.2, el desarrollo está organizado en *sprint*s siguiendo las guías de Scrum. En la Sección 9.3 entraremos en más detalle y veremos qué clases se implementan en cada *sprint*, y los motivos que han llevado necesitar dichas clases.

9.1 Módulos

Hemos dividido la lógica de la aplicación en diferentes módulos, cada uno de los cuales agrupa las clases funcional o semánticamente relacionadas. Dado que la implementación es en Java, cada uno de los módulos es en realidad un `package`. Son los siguientes[1]:

`package com.indra.iquality.controller` Clases que representan controladores.

[1]Para más información sobre cada módulo y las clases que lo forman, véase el Apéndice B.

package com.indra.iquality.dao Clases (interfaces) para abstraer el acceso a la capa de persistencia (ver Subsección 8.2.3). Determinan los métodos que la aplicación necesita independientemente de la base de datos. Al ser interfaces **deberán** ser implementadas por otras clases para ser utilizadas.

package com.indra.iquality.dao.jdbctemplate Clases que implementan los DAOs del módulo anterior para un caso particular: acceso a una base de datos Oracle mediante la libreria `JDBCTemplate` de Spring.

package com.indra.iquality.helper Clases que implementan funcionalidades auxiliares que pueden ser útiles para cualquier módulo. Por ejemplo, filtrar valores nulos obtenidos de la base de datos, o manipular árboles N-arios (usado para el árbol del diccionario de conceptos).

package com.indra.iquality.model Clases del modelo de la aplicación. Son las entidades de la lógica de negocio.

package com.indra.iquality.singleton Contiene un *singleton* con parámetros que precisamos sean homogéneos en toda la aplicación. Hemos considerado que esta clase merecía estar en un paquete separado dado que conceptualmente no es lo mismo que una clase del modelo, y en un futuro puede interesar tener más *singleton*s.

9.2 Modelo de testeado

En cuanto al testeado de la aplicación, en un principio se consideró seguir metodologías muy populares en el mundo *agile*, a saber: Test Driven Development (TDD) o Behaviour Driven Development (BDD). Aunque no son metodologías de gran complejidad y he tenido un somero contacto con ellas, es cierto que no estoy suficientemente familiarizado con ninguna como para utilizarla con soltura. Adquirir el nivel necesario para aplicarlas al proyecto y dominar *frameworks* como Cucumber hubiera supuesto una complejidad añadida a las ya existentes. Es por esto que no nos pareció necesario correr el riesgo ni invertir tiempo en esto, y descartamos estas opciones en las reuniones iniciales.

Por lo tanto, el enfoque que adoptamos para el testeado es, tras cada *sprint*, crear tests unitarios para las clases desarrolladas durante el mismo. Usamos para ello el *framework* jUnit.

Ahora bien, el testeado no incluye, en general, las clases del modelo, ya que prácticamente los únicos métodos que implementan son *getters* y *setters*. Sin embargo, sí que cobran mucha importancia los tests hechos sobre las clases que implementan el acceso a la base de datos. Como sabemos los resultados esperados de ellos[2], ya que podemos ejecutar las *queries* en el

[2]Recuérdese que la capa de datos no ha sido modificada para el proyecto, y por lo tanto podemos confiar en los resultados obtenidos, tomando como premisa que la versión anterior de la aplicación fue implementada correctamente.

SQLDeveloper, usamos estos valores para los tests.

9.3 Sprints

A continuación presentamos una descripción detallada de los módulos, clases y métodos implementados en cada *sprint*, además de los procesos seguidos para testearlos. Para más información sobre cualquier método o clase de los que se nombran a continuación, se puede consultar el Apéndice B.

9.3.1 Sprint 1

Desarrollo

Esta es la primera toma de contacto con Spring. Los pasos seguidos es esta etapa son:

- Descargar y configurar STS.
- Descargar maven y configurar tras el *proxy* de Indra.
- Crear un repositorio privado en GitHub y configurar git tras el *proxy* de Indra.
- Empezamos a construir la aplicación: configurar el `DispatcherServlet` en los ficheros `web.xml` y `mvc-dispatcher-servlet.xml`.
- Implementar el controlador que servirá las páginas genéricas de inicio, inicio de sesión, error, etc.: `public class BaseController`.
- Configurar la fuente de datos de los DAOs y el *driver* de Oracle en `spring.xml`.
- Para conectarnos al Data mart y obtener los primeros datos, implementamos las clases: `public class Flow`, `public class FlowDAO` y `public class FlowDAOJDBCTemplateImpl`, y el controlador `public class FlowManagementController`.
- Implementar una primera versión del *frontend* muy sencilla, para poder ver los datos obtenidos.

Testing

Lo que queremos comprobar en esta primera toma de contacto es si la aplicación arranca y funciona, y si obtenemos los datos correctos de la base de datos.

La primera parte la comprobamos manualmente: arrancamos la aplicación y comprobamos si está funcionando en `localhost:8080`. Seguimos los pocos enlaces creados para ver que nos llevan a donde esperamos.

La segunda parte la comprobamos de la siguiente manera:

- Nos conectamos al Data mart mediante el SQLDeveloper. Lanzamos una query para obtener todos los pases.
- Creamos un test para comprobar que la aplicación obtiene los mismos valores. En concreto: que el número de filas obtenidas es el mismo, que el primer *record* tiene el mismo identificador, nombre y fecha que el obtenido manualmente, y lo mismo para el último *record*.
- Comprobamos visualmente si los datos mostrados en la vista son los mismos que vemos en el SQLDeveloper para la misma *query*.

9.3.2 Sprint 2

Desarrollo

Dado que en el *sprint* anterior conseguimos conectarnos a la base de datos, en esta etapa nos centramos en mejorar la interfaz e implementar una primera funcionalidad. Los pasos seguidos son:

- Buscar y elegir un *template* de *frontend* que use Bootstrap y jQuery.
- Integrar el *template* a la aplicación.
- Crear pantallas para ver ejecuciones, jobs de una ejecución y registro de operaciones de un job.
- Implementar las clases y métodos necesarios para obtener los datos de ejecuciones, jobs y registro de operaciones, a saber:
  - Las clases del modelo: `public class Execution`, `public class Job`, `public class`

`RegisterOfOperation`, `public class Dependency` y `public class OperationTrace`.

- `public class ExecutionManagementController@showAllExecutions`
- `public class ExecutionManagementController@getJobsOfExecution`
- `public class ExecutionManagementController@getRegopsOfJob`
- `public class APIController@getJobDependencies`
- `public class APIController@getOperationTrace`
- `public interface ExecutionDAO@getAll` y su implementación en `public class ExecutionDAOJDBCTemplateImpll`
- `public interface JobDAO@getAllOfExecution` y su implementación en `public class JobDAOJDBCTemplateImpl`
- `public interface RegisterOfOperationsDAO@getAll` y su implementación en `public class RegisterOfOperationsDAOJDBCTemplateImpl`
- `public interface TraceOfRegisterDAO@getAll` y su implementación en `public class TraceOfRegisterDAOJDBCTemplateImpl`
- `public interface DependencyDAO@getAll` y su implementación en `public class DependencyDAOJDBCTemplateImpl`

Testing

Nuevamente, tenemos que comprobar que los datos obtenidos por los DAOs son correctos, y que se muestran correctamente en las vistas correspondientes. Para ello:

- Lanzamos las mismas queries que los DAOs en el SQLDeveloper. Con los datos obtenidos, creamos tests unitarios para cada uno de los DAOs enumerados en el apartado anterior y comprobamos: que el número de filas obtenidas es el mismo, que el primer *record* tiene el mismo identificador, nombre y otros campos significativos que el obtenido manualmente, y lo mismo para el último *record*.
- Para comprobar que los datos se muestran por pantalla correctamente:
  - Comprobamos que aparecen datos en la pantalla cuando seleccionamos la opción correspondiente.

- Comprobamos que estos datos se corresponden con los obtenidos mediante el SQL-Developer.

- Comprobamos que estos datos se corresponden con los que se muestran en la versión anterior de *iQuality*.

9.3.3 Sprints 3 y 4

Dado que estos dos *sprint*s están muy relacionados, los detallaremos conjuntamente.

Desarrollo

En esta etapa nos centramos en las funcionalidades del árbol de conceptos. Dado que conseguir utilizar el *plug-in jTree* (véase la Subsección 8.3.1) ya presenta en sí cierta complejidad, dividimos esta etapa en tres partes:

- Crear la pantalla del diccionario con un árbol de prueba cualquiera

- Obtener el árbol y los conceptos del Data mart. Para eso implementamos:

  - Las clases del modelo: `public class DictionaryConcept`, `public class AttributeDescription`, `public class MasterAttributeDescription` y `public class IndicatorDescription`.

  - `public class APIController@getComponentDescription`

  - `public class APIController@getDictionaryAsJSONTree`

  - `public class APIController@getComponentDescription`

  - `public class DictionaryController@showDictionary`

  - `public interface DictionaryOfConceptsDAO@getAllConcepts` y su implementación en `public class DictionaryOfConceptsDAOJDBCTemplateImpl`

  - `public interface DictionaryOfConceptsDAO@getDescriptionOfAttribute` y su implementación en `public class DictionaryOfConceptsDAOJDBCTemplateImpl`

  - `public interface DictionaryOfConceptsDAO@getDescriptionOfMasterAttribute` y su implementación en `public class DictionaryOfConceptsDAOJDBCTemplateImpl`

  - `public interface DictionaryOfConceptsDAO@getDescriptionOfIndicator` y su implementación en `public class DictionaryOfConceptsDAOJDBCTemplateImpl`

- Usar los datos obtenidos en vez de los de prueba

Dado que obtener los conceptos y generar el árbol del diccionario cada vez es muy costoso temporalmente[3], el enfoque que hemos adoptado es: generar el diccionario una vez, *parsearlo* a JSON en el formato esperado por *jTree*, y guardar este JSON en un fichero que hace de caché. De esta manera, cada vez que carguemos la pantalla del diccionario sólo será necesario leer el árbol del fichero. Finalmente, será necesario disponer de un botón mediante el cual poder actualizar el fichero cuando introduzcamos un nuevo concepto en el diccionario (cosa que no es muy frecuente). Estas funcionalidades se cubren implementado:

- La clase auxiliar para representar los nodos del árbol `public class GenericTreeNode`.

- La clase auxiliar para transformar los *records* obtenidos de la *query* a la representación JSON que espera *jTree*: `public class TreeTranslatorHelper@conceptListToTree`, `public class TreeTranslatorHelper@treeToJSONStringForjsTree` y `public class TreeTranslatorHelper@treeToPrettyJSONStringForjsTree`. La complejidad de la transformación reside en el primer método, que implementa un algoritmo *ad hoc* a partir de la *query* de conceptos.

- Los métodos para actualizar la caché: `public class APIController@updateDictionaryCache` y `public class APIController@auxiliaryUpdateDictionaryCache`

Testing

Se trata ahora de comprobar que las partes anteriores funcionan como es esperado:

- Para comprobar que el árbol de prueba se comporta correctamente, no hay forma más fácil que probándolo manualmente: vemos si los nodos se despliegan y contraen correctamente, si el contenido de los nodos es el esperado, si vemos conceptos la seleccionar un nodo, etc.

- Para comprobar que obtenemos una lista correcta de los conceptos y sus descripciones, seguiremos el mismo método que en las pruebas anteriores: mediante test unitarios sobre los métodos de los DAOs.

- Para constatar que el algoritmo funciona correctamente, lo comprobaremos sobre un árbol pequeño de prueba. A continuación aplicaremos el algoritmo sobre el árbol real, imprimiremos por la consola de STS el árbol y comprobaremos visualmente que los nodos están anidados como el árbol que vemos en la versión actual de *iQuality*.

[3] La pantalla del diccionario tarda del orden de 10 o 15 segundos en cargar en la versión anterior de *iQuality*, ya que lanza la *query* cada vez que entramos en esta pantalla.

- Para comprobar que el traductor de árbol a JSON funciona correctamente, imprimiremos el árbol en los dos formatos y comprobaremos que la jerarquía se corresponde.

- Para comprobar que la caché se actualiza correctamente basta con ver si el fichero se ha creado, abrirlo y comparar sus contenidos con el árbol impreso en formato JSON en la consola.

- Finalmente, para comprobar que el árbol real se integra en la vista, lo haremos de la misma manera que el primer punto del *testing*.

9.3.4 Sprint 5

Desarrollo

Llegamos a la etapa relacionada con las certificaciones: consultarlas y crearlas[4]. Los métodos involucrados en esta parte de la implementación son:

- Las clases del modelo: `public class BusinessCertificate`, `public class BusinessCertificateDetail`, `public class TechnicalCertificate` y `public class TechnicalCertificateDetail`.

- `public interface BusinessCertificateDAO@getAll` y su implementación en `public class BusinessCertificateDAO`.

- `public interface BusinessCertificateDAO@getCertificateDetails` y su implementación en `public class BusinessCertificateDAO`.

- `public interface BusinessCertificateDAO@getDetailHeaders` y su implementación en `public class BusinessCertificateDAO`.

- `public interface BusinessCertificateDAO@getCertificateConditions` y su implementación en `public class BusinessCertificateDAO`.

Testing

Siguiendo la misma metodología utilizada hasta ahora para *testear* el acceso a los datos, crearemos test unitarios y comprobaremos que los resultados obtenidos, tanto de las certificaciones como de sus datos detallados, concuerdan con los del SQLDeveloper.

[4]En el momento de escribir esto, la vista con el formulario para crear una nueva certificación está casi acabada, pero la funcionalidad de insertar una nueva certificación en la base de datos está aún por completarse. Por lo tanto, no se mencionana continuación los métodos involucrados en dicha tarea.

9.3.5 Sprint 6

Desarrollo

Llegamos finalmente a la etapa de implementación de las funcionalidades de Excel y gestión de usuarios. Ahora bien, como ya habíamos advertido en la Sección 3.1, la gestión de usuarios no es un punto de gran importancia actualmente, por lo tanto hemos implementado la interfaz (véase la Sección 8.4.2) pero no la funcionalidad. Por otra parte, al vernos apremiados por el tiempo al llegar al último *sprint*, las funcionalidades relativas al manejo de ficheros Excel se han visto afectadas: hemos descartado la funcionalidad de exportado, y la de importado aún no inserta datos en una tabla. Sí hemos sido capaces, sin embargo, de subir ficheros Excel al servidor y leer los datos mediante una librería de Apache (véase la Subsección 8.3.2). Dicho esto, los métodos y clases asociados a estas funcionalidades son:

1. `public class BaseController@showLogin`.
2. `public class BaseController@showRegistration`.
3. `public class FileLoadController@loadFiles`.
4. `public class FileLoadController@handleFileUpload`.

Testing

En cuanto a la gestión de usuarios, lo único que podemos *testear* hasta el momento es el comportamiento de la interfaz, que lo haremos manualmente.

La comprobación de la lectura de archivos Excel, en cambio, la podemos automatizar con un test unitario. Crearemos un archivo Excel con campos diversos, lo subiremos y el test comprobará si la aplicación lee el fichero, y si los valores leídos son los esperados.

9.3.6 Resumen

Siguiendo el hilo de los *sprint*s, hemos visto en qué momento y a partir de qué necesidades nace cada método y clase de la implementación. Es importante tener esta visión histórica, para comprobar que ciertamente cada parte del código está respaldada por algún requisito de la aplicación.

Hemos justificado también el proceso de *testeado* seguido en cada *sprint*, de forma que podemos defender y tener cierta seguridad de la robustez de la aplicación.

Finalmente, es conveniente comentar que algunas clases de la implementación no han sido nombradas en ninguno de los *sprints* por el simple hecho de ser de carácter tan global que son igual de importantes en todo momento. Nos estamos refiriendo principalmente a las clases `public class Environment` y `public class DataHelper`, así como a `public enum StatusEnum` y `public enum ConceptTypeEnum`. Se puede leer más sobre ellas en el Apéndice B.

9.4 Modelo de despliegue

El modelo de despliegue muestra todos los componentes del sistema en el entorno de producción, así como la relación que existe entre ellos. Como ya habíamos explicado brevemente en la Sección 1.2, *iQuality* consta de un Data mart, un servidor Unix y una aplicación web. El modelo de despliegue en producción será el mismo que en la versión anterior, con la diferencia de que cambia la aplicación web. Lo podemos ver a continuación:

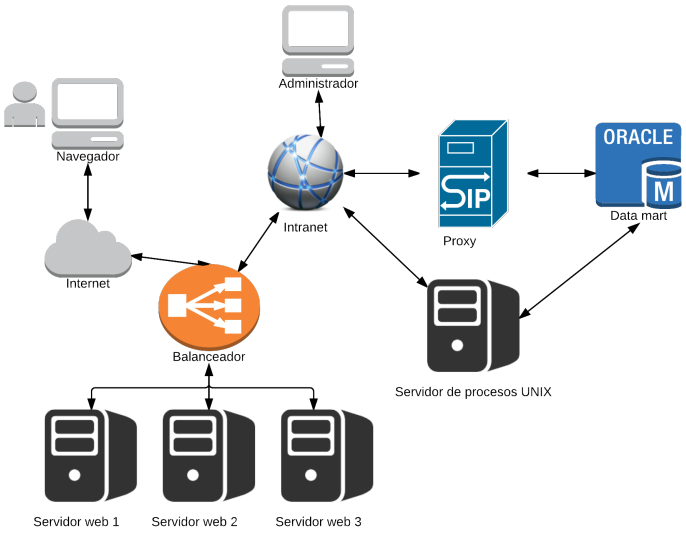

Figura 9.1: Modelo de despliegue de *iQuality* en producción.

Podemos distinguir las tres partes: Data mart, servidor de procesos y servidor web. Cuando la aplicación se comience a usar con frecuencia, nos interesa tener tres réplicas del servidor tener más tolerancia a fallos y más eficiencia[5], equilibrados mediante un balanceador de carga. El usuario se conectará a la aplicación mediante un navegador a través de internet, y los servidores obtendrán datos del Data mart, protegido tras un *proxy* con cortafuegos. Además, el administrador se puede conectar al servidor de procesos Unix a través de la intranet corporativa, y lanzar procesos directamente sobre el Data mart.

[5] Aunque, en última instancia, está claro que el cuello de botella vendrá dado por la conexión con el Data mart.

10 Conclusiones

Llegados al final del proyecto, es momento de preguntarnos si hemos conseguido alcanzar los objetivos planteados. Además, es importante dejar un pequeño plan de ruta para guiar el desarrollo de las siguientes etapas y versiones de *iQuality*.

10.1 Conclusiones

Si tuviese que resumir las conclusiones del proyecto en dos palabras, diría: *Éxito, pero...*

Recapitulando los objetivos planteados en la Sección 3.1, podemos afirmar que hemos sido capaces de migrar *iQuality* a una implementación plenamente funcional utilizando las tecnologías elegidas. *Pero* no es menos cierto que algunas pantallas no disponen aún de todas las funcionalidades de la versión antigua del *software*: podemos crear y consultar pases pero no crearlos, no podemos importar ficheros Excel, etc. (Se detallan estas carencias en la siguiente sección). Sin embargo, esto no menoscaba el hecho de que el núcleo de cada una de las funcionalidades que planteamos dentro del alcance del proyecto ha sido resuelto e implementado exitosamente.

Además, hemos conseguido mejorar los aspectos de la versión anterior que considerábamos deficientes:

- La usabilidad y el *look & feel* se han enriquecido notablemente, como se ha visto en la Subsección 8.4.2 y la Subsección 8.4.1.

- El tiempo de respuesta ha disminuido muchísimo, especialmente en cuanto al diccionario de conceptos: el árbol ha pasado de tardar 20 segundos en cargar a apenas 2 o 3. Más notablemente aún, los detalles de los conceptos, que antes ni tan siquiera se podían cargar, se pueden visualizar ahora de manera casi instantánea.

- Mantener y entender la aplicación resulta ahora mucho más sencillo, dado que se han separado todos los componentes siguiendo un diseño arquitectónico, y disponemos de documentación sobre la implementación, principalmente el Javadoc del Apéndice B.

- Al poder empaquetar la aplicación en un archivo ejecutable y haberla desacoplado de la implementación de la base de datos, la portabilidad se ha visto muy beneficiada.

- Finalmente, hemos creado una aplicación más robusta gracias a las pruebas a las que la hemos sometido, y el exhaustivo registro de *log*s del que disponemos para rastrear las acciones de la misma.

Por lo tanto, podemos afirmar con seguridad que gracias a este proyecto hemos producido unos sólidos y estructurados fundamentos para la nueva versión de *iQuality*, sobre los que resultará sencillo seguir edificando la herramienta. Aunque los objetivos iniciales eran un poco más ambiciosos, al intentar migrar prácticamente la totalidad del *software*, las desviaciones experimentadas durante el transcurso del proyecto fueron acordadas y aceptadas por los *stakeholders*, y se han mostrado satisfechos con los resultados obtenidos. Por otra parte, las horas dedicadas y el resultado conseguido cubren con creces la carga de trabajo esperada de un trabajo de final de grado.

10.2 Tareas futuras

No podemos terminar el proyecto sin contemplar los próximos pasos. Estos incluyen tanto facetas que quedaban fuera del proyecto desde el planteamiento inicial, como otras que caían dentro del alcance pero no fuimos capaces de finalizar en el término establecido. Como la idea es que esta pequeña guía resulte útil tanto para plantear las próximas iteraciones como para contemplar aspectos detallados de la implementación, la dividiremos en dos partes: aspectos generales y aspectos específicos, que pueden ser desde propuestas de mejora hasta *bugs* conocidos.

10.2.1 Tareas generales

- Contemplar la disponibilidad y durabilidad de la aplicación.
- Contemplar la escalabilidad de la aplicación,
- Contemplar la seguridad de la aplicación.
- Acabar de implementar la importación y exportación de ficheros Excel.
- Implementar la gestión de usuarios, el inicio de sesión y el registro.
- Implementar la gestión de perfiles y otros aspectos administrativos.
- Acabar de implementar la pantalla inicial, tanto la interfaz como el uso de datos reales.
- Hay dos opciones del menú que no hemos incluido: gestión de certificaciones y gestión de dimensiones.

- Falta la funcionalidad de las alarmas por fallos de ejecuciones.

10.2.2 Tareas específicas

- Refactorizar el *frontend*: comentar el código, agrupar funcionalidades comunes, eliminar librerías y dependencias que no se usan.

- Refactorizar el *backend*: optimizar *queries* para que devuelvan sólo los campos que necesitamos, eliminar atributos innecesarios de algunas clases.

- Mejorar la interfaz del *wizard* para crear un nuevo pase; en concreto, la parte donde seleccionamos las dependencias de los jobs.

- En el *wizard* para crear nuevos pases no se contempla que se pueden formar bucles entre las dependencias. La base de datos también permite cometer este error.

- En la gestión de pases, cada pase sale triplicado para tres versiones: inicial, inicial-met y prueba.

- Falta el campo "métrica evaluada" en el resultado de las certificaciones de negocio.

- Comprobar que todas las *queries* tienen en cuenta el identificador del sistema y la versión de software.

- Separar los tipos de certificaciones en pestañas más estéticas.

- Considerar si sería mejor cambiar los campos binarios que están implementados como *strings* a booleanos.

Glosario

Agile Engloba un conjunto de métodos de desarrollo de *software* en el cual los requerimientos y soluciones evolucionan mediante la colaboración de miembros de un equipo autogestionado y multidisciplinar. 74, 90, *véase* Scrum

AJAX (también **Ajax**) Asynchronous JavaScript and XML. Es un conjunto de técnicas de desarrollo web utilizadas del lado del cliente para crear aplicaciones web asíncronas, i.e., para interactuar con el servidor en el *background*, sin necesidad de recargar la página. 11

Apache Maven Es una herramienta para gestionar las dependencias de un proyecto de *software*. Permite automatizar esta gestión en base a especificar las dependencias, y extraerlas a partir de un repositorio centralizado para incluirlas en la construcción de nuestro proyecto. 15, 18, 21, 24, 74

Apache Tomcat Es una implementación gratuita de diferentes tecnologías estándar de servidores Java. Por lo tanto, es un *software* que nos permite utilizar un ordenador como servidor, ya se a nivel local para hacer pruebas o para poner una aplicación en producción. 21

APEX APplication EXpress. *Framework* propietario de Oracle para desarrollar aplicaciones web con pocos conocimientos de tecnologías web mediante un navegador. 10, 11, *véase* *framework*

API Application Programming Interface. Expresa un componente *software* en término de sus operaciones, entradas y salidas, definiendo las funcionalidades del sistema independientemente de su implementación. 43

Back end (también **back-end, backend**) Se refiere a la infraestructura que hay detrás de una aplicación: mínimamente, una base de datos y un servidor. El término se utiliza sobre todo para aplicaciones web, y en contraposición a *frontend*. 10, 15, 18, 24, 39, 40, 43, 85, 88

BDD Behaviour Driven Development. Esta metodología de desarrollo emerge a partir de TDD, pero poniendo el foco del desarrollo sobre los comportamientos del *software*, así como la implicación tanto del equipo de gestión como del cliente en el proceso de desarrollo. 74

BI Business Intelligence. Engloba el conjunto de técnicas y herramientas mediante las cuales se puede extraer información significativa a partir de datos puros, con tal de usarla en el

análisis y gestión de una organización. 20

BPMN Business Process Model and Notation. Es un modelo de representación gráfica para especificar procesos de negocio. 62

Centro de Procesamiento de Datos (CPD) Es la ubicación donde se encuentran los recursos necesarios para el procesamiento de la información (datos) de una organización. 26, 29

Certifiación En el contexto de *iQuality*, representan reglas de calidad de datos que se pueden aplicar sobre tablas o conjuntos de tablas. En general, son los usuarios los que las crean conforme a las reglas de negocio, y se utilizan para comprobar la calidad de los datos. 4, 16, 18, 25, 79

CSS Cascading Style Sheets. Es un lenguaje para describir la presentación de un documento escrito en HTML. 40, 42

Data mart Es la capa de acceso a un almacén de datos (*data warehouse*) que sirve para que los usuarios accedan a los datos. 3–6, 12, 21, 26, 27, 29, 41, 71, 75, 77, 81, 82

Data quality Como su nombre indica, se refiere al nivel de calidad de los datos. Cómo se mide esta calidad depende mucho de cada organización y de cada entorno. En la Sección 1.1 se habla extensamente sobre este tema, citando los parámetros de calidad más habituales. 7, 9

Dependencia En el contexto de *iQuality*, las dependencias representan las relaciones de precedencia entre diversos jobs. Decimos que un job *depende* de otro si este debe ejecutarse necesariamente antes que aquél, y viceversa. 4, 11, *véase* job

Device driver Es un programa encapsulado que controla un tipo de dispositivo particular conectado a algún sistema informático. Por ejemplo, hay *drivers* para conectar un programa a una base de datos. 11, 75

Diccionario de conceptos En el contexto de *iQuality*, es la representación de todos los conceptos de negocio de VidaCaixa, a saber: entidades, indicadores, cálculos, etc. La representación del diccionario sigue una jerarquía representada por un árbol, donde los nodos terminales son los conceptos. 11

Ejecución En el contexto de *iQuality*, representa una instancia de un pase en un día y sistemas concretos. 4, 16, 18, 25, 76, 90

ETL Extraction, Transformation and Load. Es el proceso de extraer, transformar y cargar datos para integrarlos en un sistema de gestión de datos. 3, 4

Fachada (también façade, front controller) Patrón arquitectónico que reduce la complejidad de un subsistema *software* y proporciona un punto de acceso común a todas sus funcionalidades. Véase la Subsección 8.2.2 para más información. 11

Framework Es un entorno *software* reusable que proporciona un conjunto de funcionalidades concretas como parte de una plataforma más grande para facilitar el desarrollo de aplicaciones, productos y soluciones *software*. 10–12, 15, 18, 21, 24, 25, 40, 41, 44, 74

Front end (también **front-end, frontend**) Se refiere a la parte de una aplicación *software* que podemos ver y con la cual interactuamos. El término se utiliza sobre todo para aplicaciones web, y en contraposición a *backend*. 10, 15, 16, 18, 21, 24–26, 28, 39, 40, 75, 76, 85, 86

Git Sistema de control de versiones ampliamente utilizado en el mundo del desarrollo de *software*. 74, 88, *véase* GitHub

GitHub Es un servicio de repositorios git que extiende las funcionalidades de la implementación original, añadiendo entre otras cosas una interfaz web e integración en ordenadores de escritorio y dispositivos móviles. 21, *véase* git

IDE Integrated Development Environment. Es una aplicación *software* que proporciona un extenso conjunto de servicios y utilidades para facilitar el desarrollo de *software* a los desarrolladores. La idea es que el programador pueda realizar todas las tareas necesarias para el desarrollo sin tener que salir de la aplicación. 21, 41

Indra Es una de las consultoras tecnológicas multinacional más prominentes. Es la impulsora y propietaria de *iQuality*. 3, 6, 9, 13, 20, 21, 24, 74

Javadoc Es un generador de documentación para Java, capaz de crear la documentación de la API de un proyecto en formato HTML a partir de comentarios en el código fuente. Por extensión, y no sin cierto abuso del lenguaje, se utiliza el mismo término para referirse también a la documentación generada. 11, 16, 17, 19, 25, *véase* API

JavaScript Lenguaje de programación de alto nivel, dinámico, no tipeado e interpretado, ampliamente utilizado para la creación de páginas web. 10, 40

JDBC Java Database Connectivity. Es una API para Java que define cómo una aplicación cliente puede acceder a una base de datos, normalmente relacionales. 42, *véase* API

Job En el contexto de *iQuality*, representa un conjunto encapsulado de operaciones necesarias para una etapa de una ETL. 4, 11, 16, 18, 25, 43, 76, 87, 90, *véase* ETL, &

jQuery Es una librería de JavaScript sumamente popular, que extiende sobre sus funcionalidades y simplifica el desarrollo del lado del cliente de una aplicación web. 10, *véase*

JavaScript

JSON JavaScript Object Notation. Es un formato estándar que usa texto que puede ser leído naturalmente por las personas para transmitir objetos de datos consistentes en pares atributo-valor. 43, 78, 79

jUnit Es un *framework* para Java que permite realizar tests unitarios automatizados muy fácilmente. 11, 21

Log Es un registro de las actividades que un sistema *software* ha llevado a cabo. Normalmente se guarda en uno o varios ficheros de texto. 32, 39, 84, 89, *véase logging*

Logback Es un *framework* para generar *logs* de forma fácil y personalizable. 11, *véase log*

Logging El hecho de dejar constancia de un *log* de las actividades que realiza el sistema, ya sea en un fichero o por pantalla. *véase log*

Look & feel Utilizado con respecto a una interfaz gráfica de usuario, describe los elementos que la componen, colores, formas, distribución (*look*), así como su comportamiento dinámico mediante botones, menús, selectores (*feel*). 10, 46, 83

Migrar En el entorno del desarrollo de *software*, significa reimplementar una aplicación o sistema existente utilizando otras tecnologías diferentes. 10, 11

MVC Modelo Vista Controlador. Es un patrón de desarrollo de *software*. Divide una aplicación (o alguno de sus submódulos) en tres partes interconectadas, de manera que podamos separar la representación interna de los datos de las formas en que la información es presentada al usuario o aceptada por el mismo. Véase la Subsección 8.1.1 para más información. 11, 41, 42

Negocio Entendido como en el inglés *bussiness*, i.e., cualquier organización o entidad, no necesariamente con ánimo de lucro. 1, 2

NoSQL Modelo de bases de datos que no utiliza el modelo tradicional de tablas relacionales para almacenar los datos. 11

Pase (también **flow**) En el contexto de *iQuality*, representa un proceso ETL. Cada pase está formado de diversos jobs, y a cada instancia de un pase la denominamos ejecución. 4, 11, 43, 83, 87, *véase* ETL

PL/SQL Procedural Language/Structured Query Language. Es una extensión del lenguaje SQL implementada por Oracle; permite declarar variables, ejecutar bucles y condiciones, etc. 3

Plug-in (también **plugin**) Es un componente *software* que añade una funcionalidad específica a un programa o sistema existente. Por ejemplo, un *plug-in* de jQuery para generar e interactuar con una vista en forma de árbol en nuestra aplicación web. 10, 40, 77, *véase* jQuery

Product backlog Consiste en una lista ordenada de los requisitos que un equipo mantiene para un producto. Puede contener requisitos funcionales, no funcionales, mejoras, arreglo de *bugs*, y en general cualquier proceso que sea necesaria llevar a cabo para finalizar el producto. 11, 13–15, 17, 19, 21, 24, 90, *véase* Scrum

Proxy Es un servidor que actúa de intermediario entre clientes y otros servidores que guardan los recursos. 82

Registro de operaciones En el contexto de *iQuality*, se refiere a todas las operaciones que un job de una ejecución realiza. Por ejemplo, comprobación de la existencia de archivos, lectura de ficheros, llamada a otros jobs, etc. 76

Responsivity (tiempo de respuesta) Describe la rapidez con la que un sistema *responde* a las interacciones del usuario u otro sistema. Decimos que un sistema es *responsive* cuando responde rápidamente. No confundir con la *responsivity* visual de una aplicación o página web. 11, 90, *véase responsivity*

Responsivity (visual) Describe la capacidad de una aplicación o página web para adaptarse a diferentes resoluciones de pantalla, de manera que los diferentes elementos se organicen y muestren de la manera más cómoda para cada tamaño. No confundir con la *responsivity* en cuanto a rapidez de respuesta. 11, 40, 90, *véase responsivity*

REST REpresentational State Transfer. Es la arquitectura subyacente a la Web. Define el conjunto de convenciones y restricciones que hacen de la Web un sistema práctico, eficiente y, en definitiva, funcional. 43

Scrum Es una metodología iterativa e incremental de desarrollo de *software* que sigue el paradigma *agile*. Destaca por su flexibilidad y su visión holística del proceso de desarrollo. 13, 14, 30, *véase agile*

Singleton Es un patrón arquitectónico propio de la programación orientada a objetos que garantiza que sólo tenemos una instancia de un objeto. Véase la Subsección 8.2.1 para más información y ejemplos. 44, 73

Spring Es el *framework* de aplicaciones web más popular para la plataforma Java. 10, 12, 15, 18, 21, 24, 41–44, 73, 74, *véase framework*

Sprint Período de tiempo durante el cual se lleva a cabo el desarrollo sobre un conjunto de tareas del *backlog*. 13, 17, 19, 20, 25, 74, 76, 77, 80, 81, 90

Sprint backlog Es un subconjunto de las tareas del *backlog* que el equipo de desarrollo llevará a cabo durante el próximo *sprint*. 13, 17, *véase* Scrum &

SQL injection Es una técnica de inyección de código, que permite a los atacantes inyectar código maligno en la base de datos de una aplicación. 12

SQLDeveloper Es una aplicación de Oracle que permite realizar tareas sobre una base de datos (también Oracle) a través de una interfaz gráfica de usuario cómoda e intuitiva. 41, 74, 75, 77, 80

STS Spring Tool Suit. Es un IDE que específico de Spring que facilita el desarrollo en dicho *framework*. 15, 18, 21, 24, 74, 79, *véase* IDE &

TDD Test Driven Development. Es una metodología de desarrollo de *software* que se basa en los tests: primero creamos las pruebas que la funcionalidad debe pasar, implementamos la funcionalidad, falla los test, y seguimos implementando hasta que los pasa. Este ciclo se repite durante todo el desarrollo. 74, 86

Template En general, una plantilla a partir de la cual crear instancias de una misma entidad. En este proyecto, utilizamos el término para referirnos a un conjunto de documentos HTML, CSS y JavaScript que proporcionan una interfaz de usuario básica para nuestra aplicación, y a partir de los cuales podemos extender y personalizar la interfaz a nuestro gusto. 15, 18, 21, 25, 26, 76, *véase* CSS &

Timeout (también **time-out**) Lapso de tiempo permitido para realizar una petición un servidor web, transcurrido el cual será cancelada. 11

Tweeter bootstrap (también **Bootstrap**) Es el *framework* más popular para desarrollar el *front end* de aplicaciones web *responsive*. 10, 76, *véase framework*, &

Unix Familia de sistemas operativos derivados del original AT&T Unix. 3, 81, 82

VidaCaixa Entidad líder en previsión social integrada en *CaixaBank*. 3, 6, 28, 87

XSS Es un tipo de vulnerabilidad de la seguridad típico de las aplicaciones web. Permite a los atacantes inyectar *scripts* del lado del cliente en páginas vistas por otros usuarios. 12

Bibliografía

[1] CRMfusion. Demandtools product details [online]. 2015. URL: https://www.crmfusion.com/demandtools/demandtools-details/ [cited 30/09/2015].

[2] DataCleaner. The premier open source data quality solution [online]. 2015. URL: http://datacleaner.org [cited 30/09/2015].

[3] Frank Dravis. Data quality strategy: a step-by-step approach. Technical report, Ninth International Conference on Information Quality, 2014.

[4] EIOPA. Solvency ii guidelines [online]. 2015. URL: https://eiopa.europa.eu/regulation-supervision/insurance/solvency-ii [cited 17/01/2016].

[5] L.P Hewlett-Packard Development Company. Hp elitebook 840 notebook pc data sheet [online]. September 2013. URL: http://www.hp.com/united-states/campaigns/elite-products/assets/elitebook840.pdf [cited 26/12/2015].

[6] IBM. Ibm data quality management [online]. URL: http://www-03.ibm.com/software/products/en/category/SWP22 [cited 30/09/2015].

[7] Informatica. Data quality software and tools [online]. 2015. URL: https://www.informatica.com/products/data-quality/informatica-data-quality.html#fbid=iTMAGuILM4i [cited 30/09/2015].

[8] Rod Johnson and Juergen Hoeller. Spring framework reference documentation [online]. 2015. URL: http://docs.spring.io/spring-framework/docs/current/spring-framework-reference/html/ [cited 28/12/2015].

[9] Joseph M. Juran. *Juran's quality handbook*. McGraw-Hill, 5 edition, 1998.

[10] David Loshin. Business performance and data quality metrics [online]. 2006. URL: http://download.101com.com/pub/TDWI/files/DataGovernance.pdf [cited 08/12/2015].

[11] John McCallum. Disk drive prices (1955-2015) [online]. Mayo 2015. URL: http://www.jcmit.com/diskprice.htm [cited 08/12/2015].

[12] Dan Meers. Rapidly delivering data quality metrics with a repeatable process:

Is your data fit? [online]. Julio 2014. URL: http://www.dataversity.net/rapidly-delivering-data-quality-metrics-repeatable-process-data-fit/ [cited 08/12/2015].

[13] Oracle. Oracle enterprise data quality product family [online]. 2014. URL: http://www.oracle.com/us/products/middleware/data-integration/enterprise-data-quality/oracle-enterprise-data-quality-ds-430148.pdf [cited 30/09/2015].

[14] Emerson Network Power. Energy logic: Reducing data center energy consumption by creating savings that cascade across systems [online]. 2009. URL: http://www.emersonnetworkpower.com/documentation/en-us/latest-thinking/edc/documents/white%20paper/energylogicreducingdatacenterenergyconsumption.pdf [cited 26/12/2015].

[15] Leonard Richardson and Sam Ruby. *RESTful Web Services*. O'Reilly, 1 edition, 2007.

[16] RingLead. Data quality apps for crm [online]. 2015. URL: https://www.ringlead.com [cited 30/09/2015].

[17] Endesa S.A. Tarifas para empresas [online]. Enero 2015. URL: https://www.endesaclientes.com/empresas/tarifa-preferente.html [cited 26/12/2015].

[18] SAP. Sap data quality management [online]. URL: http://www.sap.com/pc/tech/enterprise-information-management/software/data-quality/index.html [cited 30/09/2015].

[19] SAS. Sas data quality [online]. URL: http://www.sas.com/en_us/software/data-management/data-quality.html [cited 30/09/2015].

[20] Experian Information Solutions. The state of data quality [online]. 2013. URL: https://www.experian.com/assets/decision-analytics/white-papers/the%20state%20of%20data%20quality.pdf [cited 08/12/2015].

[21] Spring team. Spring framework guides [online]. 2016.

[22] VidaCaixa. Vidacaixa: información corporativa [online]. 2014. URL: https://www.vidacaixa.es/es/informacion-corporativa. [cited 22/09/15].

Apéndice A Detalle de los diagramas UML

Ante la dificultad de visualizar los diagramas UML de la Sección 8.5 y Sección 8.6 en el espacio disponible de una página, presentamos a continuación una vista detallada de los diagramas introducidos en dichas secciones. Para visualizarlos mejor, los diagramas han sido divididos en sectores de la manera que se indica a continuación:

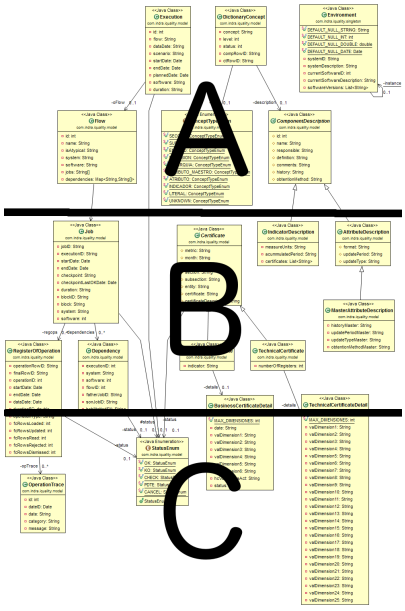

Figura 1: Subdivisión en sectores del diagrama UML de las entidades del modelo.

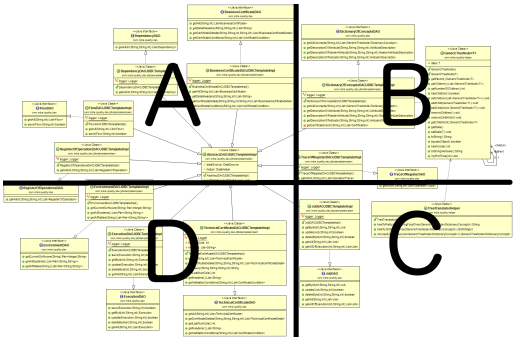

Figura 2: Subdivisión en sectores del diagrama UML de las interfaces de datos.

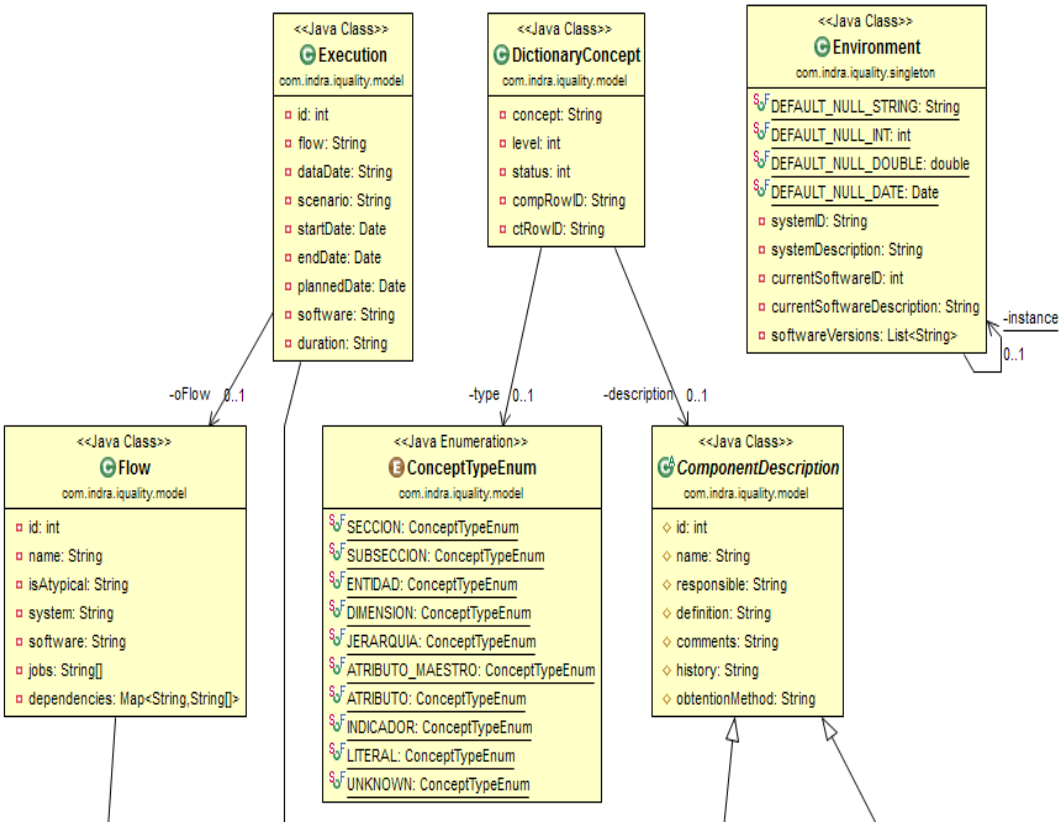

Figura 3: Detalle del sector A del diagrama UML de las entidades del modelo.

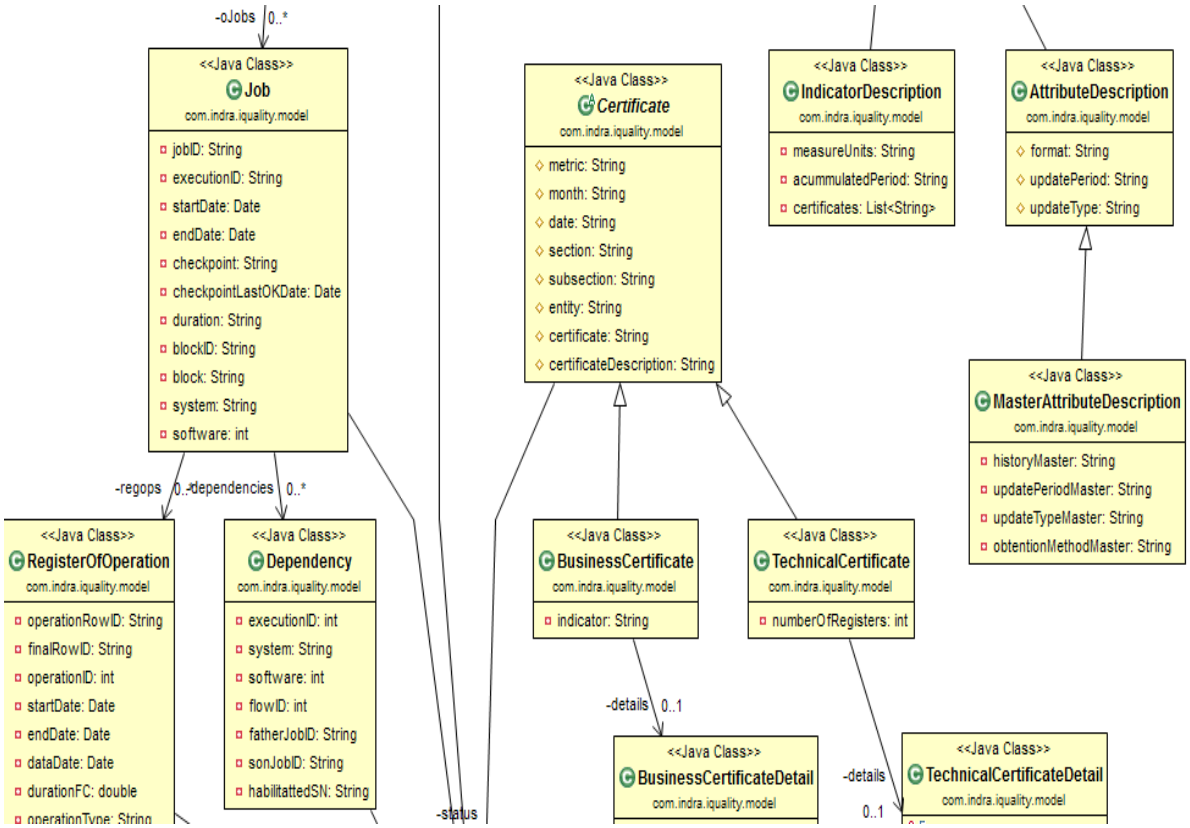

Figura 4: Detalle del sector B del diagrama UML de las entidades del modelo.

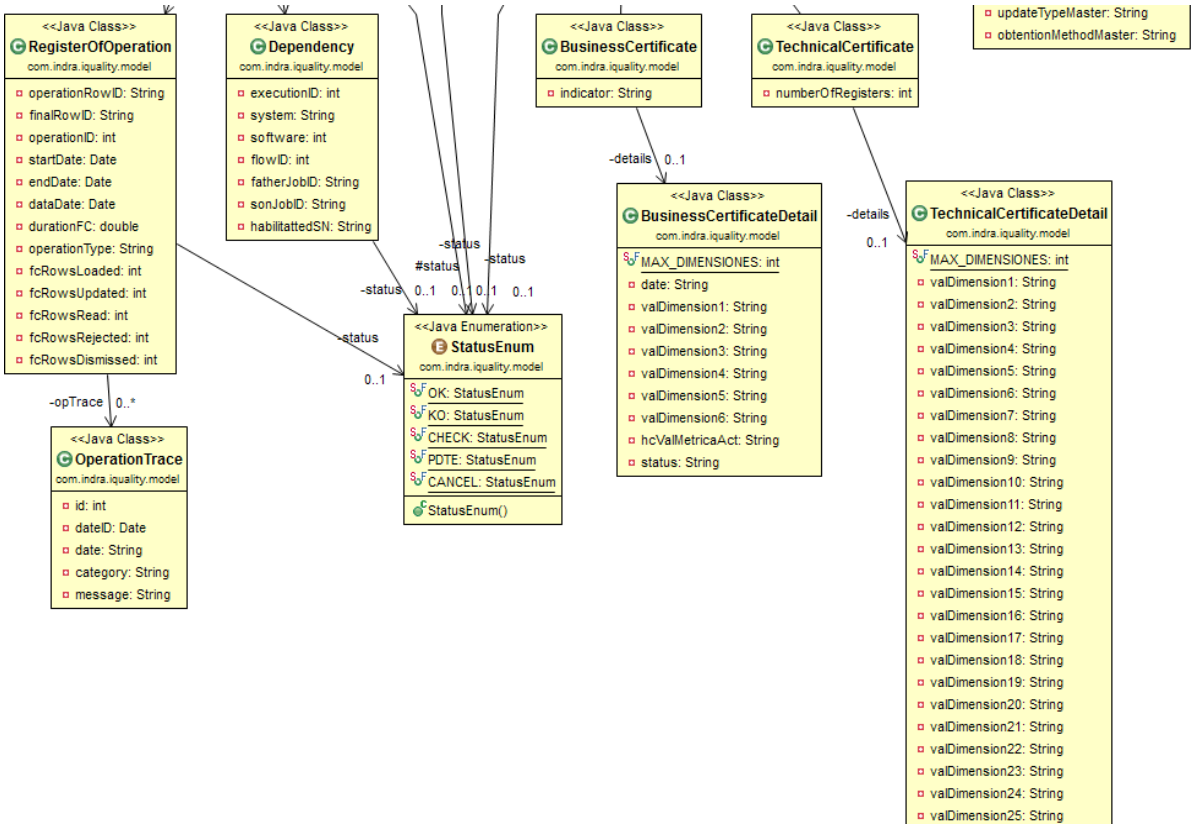

Figura 5: Detalle del sector C del diagrama UML de las entidades del modelo.

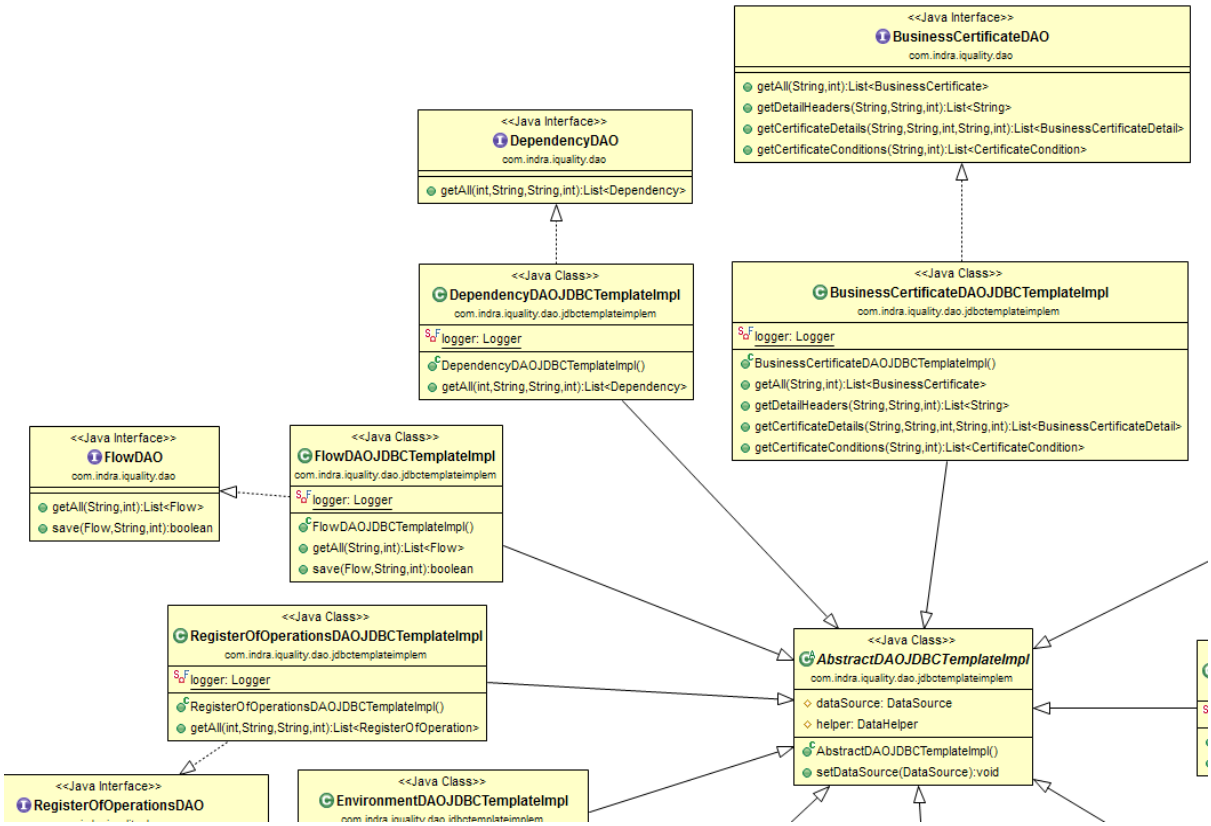

Figura 6: Detalle del sector A del diagrama UML de las interfaces de datos.

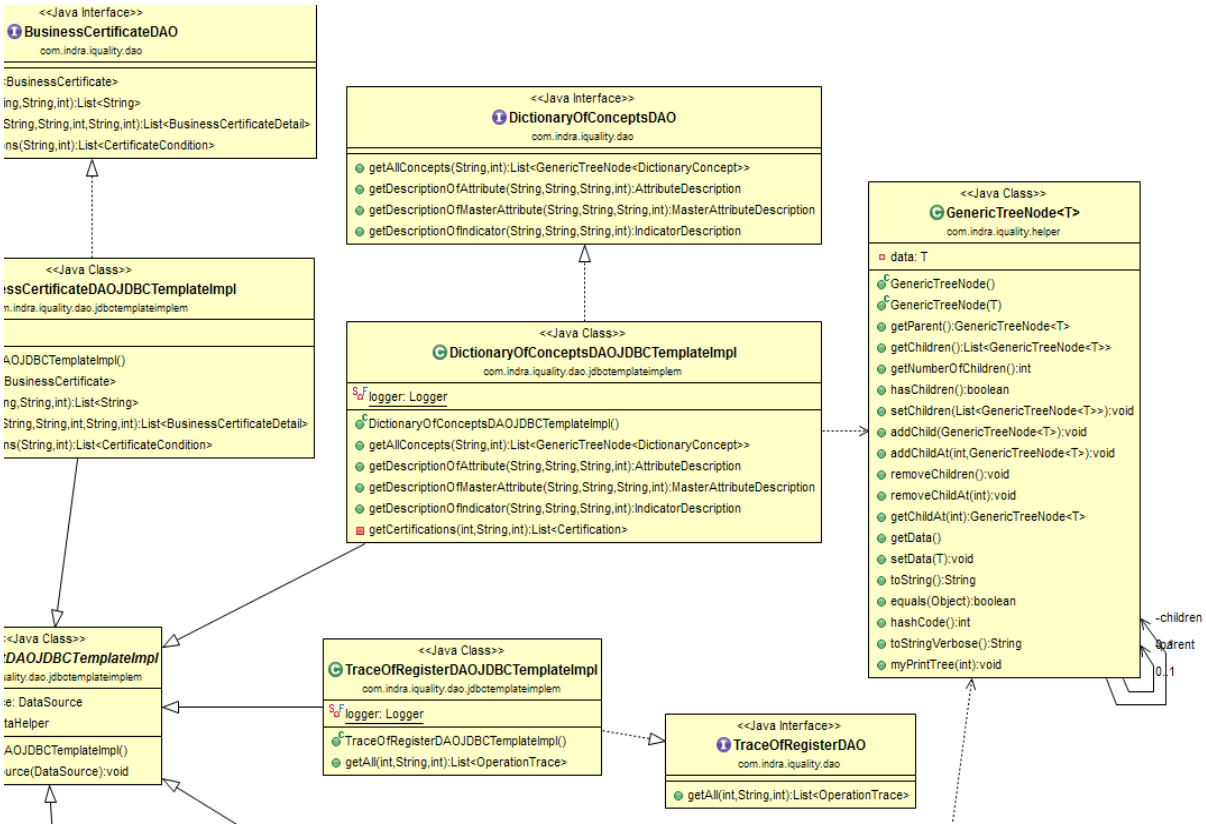

Figura 7: Detalle del sector B del diagrama UML de las interfaces de datos.

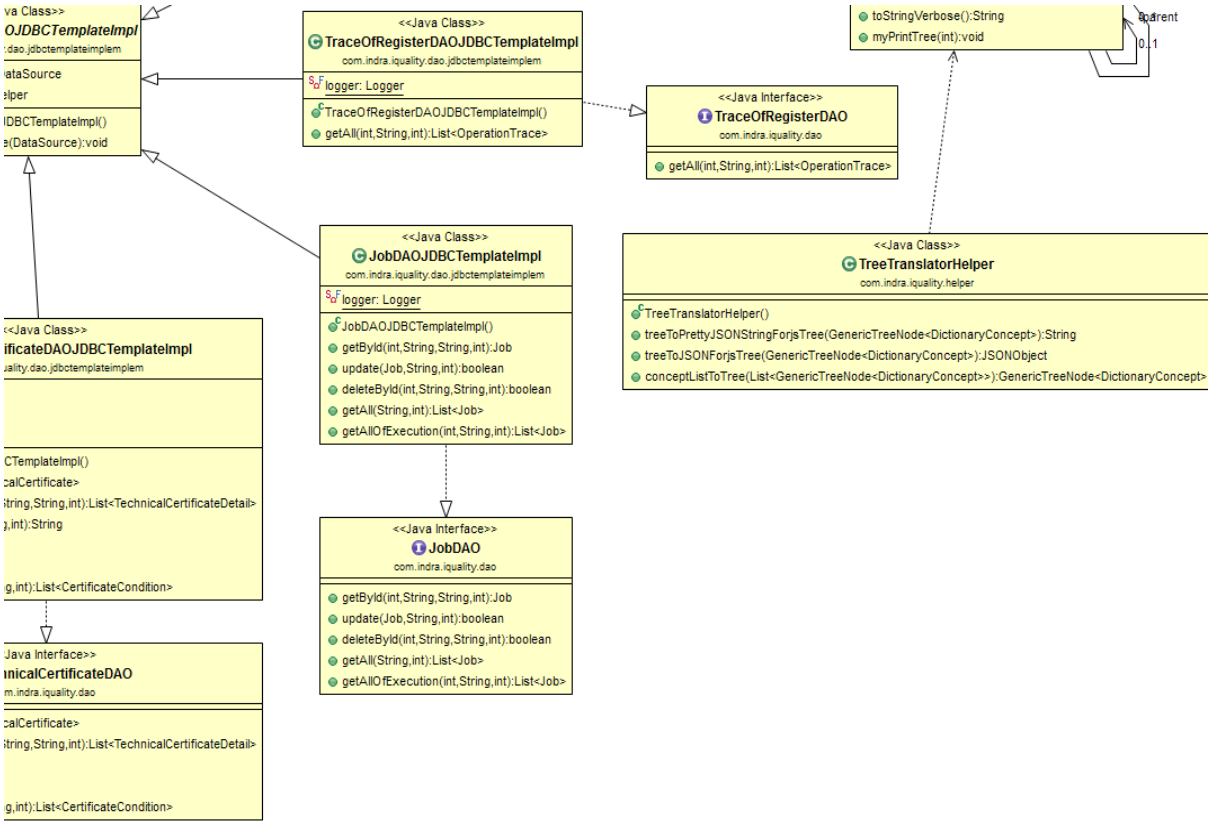

Figura 8: Detalle del sector C del diagrama UML de las interfaces de datos.

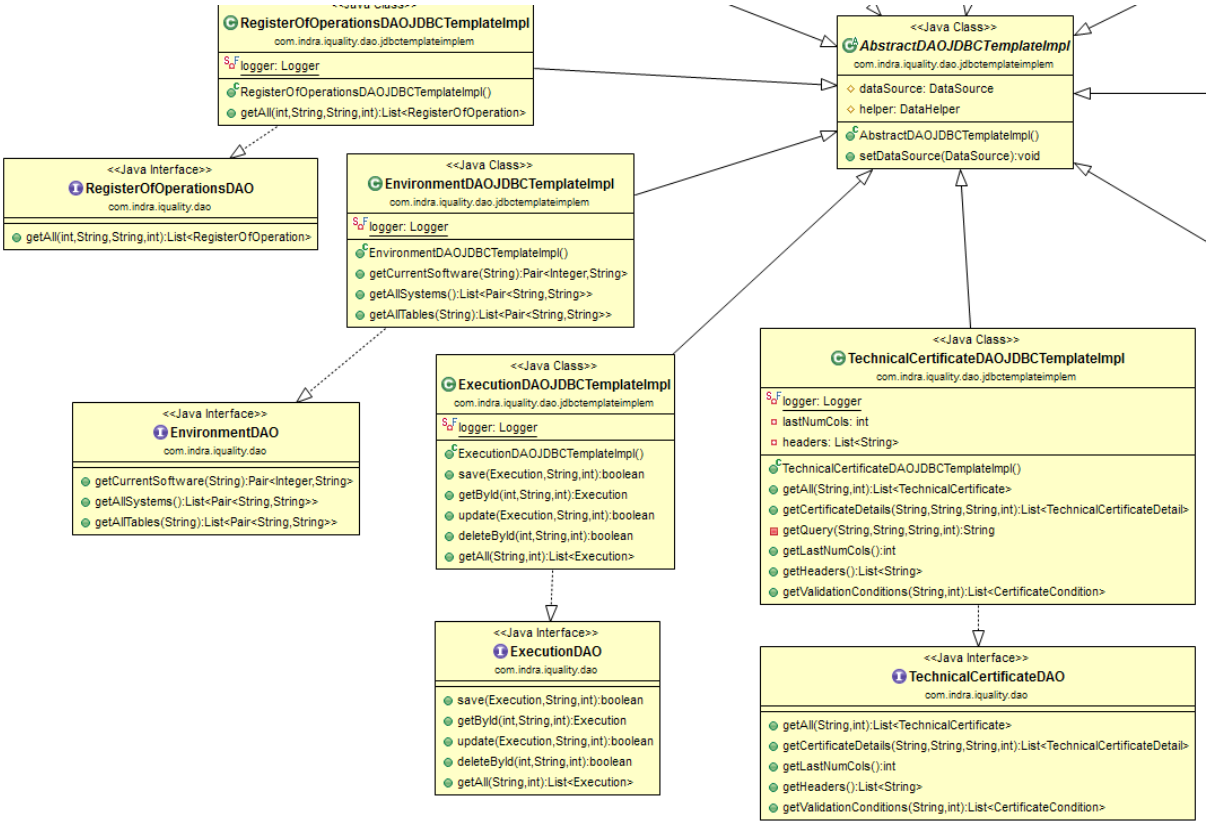

Figura 9: Detalle del sector D del diagrama UML de las interfaces de datos.

Apéndice B Javadoc

A continuación se detallan todas las clases de objetos creadas para la aplicación, así como los diferentes módulos en que se agrupan. Esto constituye la documentación técnica del proyecto que se ha generado para Indra. Debe considerarse como un apéndice al margen de la memoria.

1 Package com.indra.iquality.model

Package Contents *Page*

Classes

Provides the classes for representing the entities of the business model.

1.1 Class AttributeDescription

The Class AttributeDescription. Represents the description of an attribute element in the dictionary of concepts.

Declaration

```
public class AttributeDescription
 extends com.indra.iquality.model.ComponentDescription
```

All known subclasses

MasterAttributeDescription (in 1.14, page 183)

Field summary

format The format.
updatePeriod The update period.
updateType The update type.

Constructor summary

AttributeDescription()

Method summary

getFormat() Gets the format.
getUpdatePeriod() Gets the update period.
getUpdateType() Gets the update type.
setFormat(String) Sets the format.
setUodatePeriod(String) Sets the uodate period.
setUpdateType(String) Sets the update type.

Fields

- `protected java.lang.String` **format**
  - The format.
- `protected java.lang.String` **updatePeriod**
  - The update period.
- `protected java.lang.String` **updateType**
  - The update type.

Constructors

- **AttributeDescription**

  ```
  public AttributeDescription()
  ```

Methods

- **getFormat**

  ```
  public java.lang.String getFormat()
  ```

- **Description**

  Gets the format.

- **Returns** – the format

- **getUpdatePeriod**

  ```
  public java.lang.String getUpdatePeriod()
  ```

  - **Description**

    Gets the update period.

  - **Returns** – the update period

- **getUpdateType**

  ```
  public java.lang.String getUpdateType()
  ```

  - **Description**

    Gets the update type.

  - **Returns** – the update type

- **setFormat**

  ```
  public void setFormat(java.lang.String format)
  ```

  - **Description**

    Sets the format.

  - **Parameters**

    * `format` – the new format

- **setUodatePeriod**

  ```
  public void setUodatePeriod(java.lang.String updatePeriod)
  ```

- **Description**

  Sets the uodate period.

- **Parameters**

  * `updatePeriod` – the new uodate period

- **setUpdateType**

  ```
  public void setUpdateType(java.lang.String updateType)
  ```

  - **Description**

    Sets the update type.

  - **Parameters**

    * `updateType` – the new update type

Members inherited from class ComponentDescription

`com.indra.iquality.model.ComponentDescription` (in 1.6, page 132)

- `protected` **comments**
- `protected` **definition**
- `public String` **getComments()**
- `public String` **getDefinition()**
- `public String` **getHistory()**
- `public int` **getId()**
- `public String` **getName()**
- `public String` **getObtentionMethod()**
- `public String` **getResponsible()**
- `protected` **history**
- `protected` **id**
- `protected` **name**
- `protected` **obtentionMethod**

- `protected` **responsible**
- `public void` **setComments**(`java.lang.String` **comments**)
- `public void` **setDefinition**(`java.lang.String` **definition**)
- `public void` **setHistory**(`java.lang.String` **history**)
- `public void` **setId**(`int` **id**)
- `public void` **setName**(`java.lang.String` **name**)
- `public void` **setObtentionMethod**(`java.lang.String` **obtentionMethod**)
- `public void` **setResponsible**(`java.lang.String` **responsible**)

1.2 Class BusinessCertificate

The Class CertificacionDeNegocio. Represents a business certificate that must be satisfied for the system to be in a valid state.

Declaration

```
public class BusinessCertificate
 extends com.indra.iquality.model.Certificate
```

Field summary

details The details of the certificate.
indicator The name of the certificate.

Constructor summary

BusinessCertificate()

Method summary

equals(Object)
getDetails() Gets the details of the certificate.
getIndicator() Gets the name of the certificate.
hashCode()
setDetails(BusinessCertificateDetail) Sets the details of the certificate.

setIndicator(String) Sets the name of the certificate.
toString()

Fields

- `private java.lang.String` **indicator**
  - The name of the certificate.
- `private BusinessCertificateDetail` **details**
  - The details of the certificate.

Constructors

- **BusinessCertificate**

  ```
  public BusinessCertificate()
  ```

Methods

- **equals**

  ```
  public boolean equals(java.lang.Object arg0)
  ```

- **getDetails**

  ```
  public BusinessCertificateDetail getDetails()
  ```

  - **Description**

    Gets the details of the certificate.

  - **Returns** – the details

- **getIndicator**

  ```
  public java.lang.String getIndicator()
  ```

- **Description**

  Gets the name of the certificate.

- **Returns** – the name of the certificate

- **hashCode**

```
public native int hashCode()
```

- **setDetails**

```
public void setDetails(BusinessCertificateDetail details)
```

  - **Description**

    Sets the details of the certificate.

  - **Parameters**

    * `details` – the new details

- **setIndicator**

```
public void setIndicator(java.lang.String indicator)
```

  - **Description**

    Sets the name of the certificate.

  - **Parameters**

    * `indicator` – the new name of the certificate

- **toString**

```
public java.lang.String toString()
```

Members inherited from class Certificate

`com.indra.iquality.model.Certificate` (in 1.4, page 122)

- `protected` **certificate**
- `protected` **certificateDescription**
- `protected` **date**
- `protected` **entity**
- `public String` **getCertificate**`()`
- `public String` **getCertificateDescription**`()`
- `public String` **getDate**`()`
- `public String` **getEntity**`()`
- `public String` **getMetric**`()`
- `public String` **getMonth**`()`
- `public String` **getSection**`()`
- `public String` **getStatus**`()`
- `public String` **getSubsection**`()`
- `protected` **metric**
- `protected` **month**
- `protected` **section**
- `public void` **setCertificate**`(java.lang.String` **certificate**`)`
- `public void` **setCertificateDescription**`(java.lang.String` **certificateDescription**`)`
- `public void` **setDate**`(java.lang.String` **date**`)`
- `public void` **setEntity**`(java.lang.String` **entity**`)`
- `public void` **setIdMetrica**`(java.lang.String` **metric**`)`
- `public void` **setMonth**`(java.lang.String` **month**`)`
- `public void` **setSection**`(java.lang.String` **section**`)`
- `public void` **setStatus**`(java.lang.String` **status**`)`
- `public void` **setSubsection**`(java.lang.String` **subsection**`)`
- `protected` **status**
- `protected` **subsection**

1.3 Class BusinessCertificateDetail

The Class DetailOfCertificate. Represents the details of a business certificate.

Declaration

```
public class BusinessCertificateDetail
 extends java.lang.Object
```

Field summary

date The date.
hcValMetricaAct The value of MetricaAct.
MAX_DIMENSIONES The maximum number of dimensions that a detail view will have.
status The status.
valDimension1 The value of the dimension 1.
valDimension2 The value of the dimension 2.
valDimension3 The value of the dimension 3.
valDimension4 The value of the dimension 4.
valDimension5 The value of the dimension 5.
valDimension6 The value of the dimension 6.

Constructor summary

BusinessCertificateDetail()

Method summary

getDate() Gets the date.
getHcValMetricaAct() Gets the value of MetricaAct.
getStatus() Gets the status.
getValDimension1() Gets the value of the dimension 1.
getValDimension2() Gets the value of the dimension 2.
getValDimension3() Gets the value of the dimension 3.
getValDimension4() Gets the value of the dimension 4.
getValDimension5() Gets the value of the dimension 5.
getValDimension6() Gets the value of the dimension 6.
setDate(String) Sets the date.

setHcValMetricaAct(String) Sets the value of MetricaAct.
setStatus(String) Sets the status.
setValDimension1(String) Sets the value of the dimension 1.
setValDimension2(String) Sets the value of the dimension 2.
setValDimension3(String) Sets the value of the dimension 3.
setValDimension4(String) Sets the value of the dimension 4.
setValDimension5(String) Sets the value of the dimension 5.
setValDimension6(String) Sets the value of the dimension 6.
toString()

Fields

- `public static final int` **MAX_DIMENSIONES**
  - The maximum number of dimensions that a detail view will have.
- `private java.lang.String` **date**
  - The date.
- `private java.lang.String` **valDimension1**
  - The value of the dimension 1.
- `private java.lang.String` **valDimension2**
  - The value of the dimension 2.
- `private java.lang.String` **valDimension3**
  - The value of the dimension 3.
- `private java.lang.String` **valDimension4**
  - The value of the dimension 4.
- `private java.lang.String` **valDimension5**
  - The value of the dimension 5.
- `private java.lang.String` **valDimension6**
  - The value of the dimension 6.
- `private java.lang.String` **hcValMetricaAct**

  - The value of MetricaAct.

- `private java.lang.String` **status**
  - The status.

Constructors

- **BusinessCertificateDetail**

  ```
  public BusinessCertificateDetail()
  ```

Methods

- **getDate**

  ```
  public java.lang.String getDate()
  ```

  - **Description**

    Gets the date.

  - **Returns** – the date

- **getHcValMetricaAct**

  ```
  public java.lang.String getHcValMetricaAct()
  ```

  - **Description**

    Gets the value of MetricaAct.

  - **Returns** – the value of MetricaAct

- **getStatus**

  ```
  public java.lang.String getStatus()
  ```

- **Description**

  Gets the status.

- **Returns** – the status

- **getValDimension1**

  ```
  public java.lang.String getValDimension1()
  ```

  - **Description**

    Gets the value of the dimension 1.

  - **Returns** – the value of the dimension 1

- **getValDimension2**

  ```
  public java.lang.String getValDimension2()
  ```

  - **Description**

    Gets the value of the dimension 2.

  - **Returns** – the value of the dimension 2

- **getValDimension3**

  ```
  public java.lang.String getValDimension3()
  ```

  - **Description**

    Gets the value of the dimension 3.

  - **Returns** – the value of the dimension 3

- **getValDimension4**

  ```
  public java.lang.String getValDimension4()
  ```

- **Description**

  Gets the value of the dimension 4.

- **Returns** – the value of the dimension 4

- **getValDimension5**

```
public java.lang.String getValDimension5()
```

  - **Description**

    Gets the value of the dimension 5.

  - **Returns** – the value of the dimension 5

- **getValDimension6**

```
public java.lang.String getValDimension6()
```

  - **Description**

    Gets the value of the dimension 6.

  - **Returns** – the value of the dimension 6

- **setDate**

```
public void setDate(java.lang.String date)
```

  - **Description**

    Sets the date.

  - **Parameters**

    * `date` – the new date

- **setHcValMetricaAct**

```
public void setHcValMetricaAct(java.lang.String hcValMetricaAct)
```

- **Description**

  Sets the value of MetricaAct.

- **Parameters**

  * `hcValMetricaAct` – the new value of MetricaAct

- **setStatus**

  ```
  public void setStatus(java.lang.String status)
  ```

  - **Description**

    Sets the status.

  - **Parameters**

    * `status` – the new status

- **setValDimension1**

  ```
  public void setValDimension1(java.lang.String valDimension1)
  ```

  - **Description**

    Sets the value of the dimension 1.

  - **Parameters**

    * `valDimension1` – the new value of the dimension 1

- **setValDimension2**

  ```
  public void setValDimension2(java.lang.String valDimension2)
  ```

  - **Description**

    Sets the value of the dimension 2.

  - **Parameters**

* `valDimension2` – the new value of the dimension 2

- **setValDimension3**

  ```
  public void setValDimension3(java.lang.String valDimension3)
  ```

  – **Description**

  Sets the value of the dimension 3.

  – **Parameters**

  * `valDimension3` – the new value of the dimension 3

- **setValDimension4**

  ```
  public void setValDimension4(java.lang.String valDimension4)
  ```

  – **Description**

  Sets the value of the dimension 4.

  – **Parameters**

  * `valDimension4` – the new value of the dimension 4

- **setValDimension5**

  ```
  public void setValDimension5(java.lang.String valDimension5)
  ```

  – **Description**

  Sets the value of the dimension 5.

  – **Parameters**

  * `valDimension5` – the new value of the dimension 5

- **setValDimension6**

  ```
  public void setValDimension6(java.lang.String valDimension6)
  ```

- **Description**

  Sets the value of the dimension 6.

- **Parameters**

  * `valDimension6` – the new value of the dimension 6

- **toString**

```
public java.lang.String toString()
```

1.4 Class Certificate

The Class Certificate. Abstract representation with the fields that a certificate must have.

Declaration

```
public abstract class Certificate
 extends java.lang.Object
```

All known subclasses

TechnicalCertificate (in 1.18, page 205), BusinessCertificate (in 1.2, page 111)

Field summary

certificate The certificate.
certificateDescription The certificate description.
date The date.
entity The entity.
metric The metric.
month The month.
section The section.
status The status.
subsection The subsection.

Constructor summary

Certificate()

Method summary

getCertificate() Gets the certificate.
getCertificateDescription() Gets the certificate description.
getDate() Gets the date.
getEntity() Gets the entity.
getMetric() Gets the metric.
getMonth() Gets the month.
getSection() Gets the section.
getStatus() Gets the status.
getSubsection() Gets the subsection.
setCertificate(String) Sets the certificate.
setCertificateDescription(String) Sets the certificate description.
setDate(String) Sets the date.
setEntity(String) Sets the entity.
setIdMetrica(String) Sets the id metrica.
setMonth(String) Sets the month.
setSection(String) Sets the section.
setStatus(String) Sets the status.
setSubsection(String) Sets the subsection.

Fields

- `protected java.lang.String` **metric**
  - The metric.
- `protected java.lang.String` **month**
  - The month.
- `protected java.lang.String` **date**
  - The date.
- `protected java.lang.String` **section**
  - The section.

- `protected java.lang.String` **subsection**
  - The subsection.
- `protected java.lang.String` **entity**
  - The entity.
- `protected java.lang.String` **certificate**
  - The certificate.
- `protected java.lang.String` **certificateDescription**
  - The certificate description.
- `protected StatusEnum` **status**
  - The status.

Constructors

- **Certificate**

  ```
  public Certificate()
  ```

Methods

- **getCertificate**

  ```
  public java.lang.String getCertificate()
  ```

  - **Description**

    Gets the certificate.

  - **Returns** – the certificate

- **getCertificateDescription**

  ```
  public java.lang.String getCertificateDescription()
  ```

- **Description**

  Gets the certificate description.

- **Returns** – the certificate description

- **getDate**

  ```
  public java.lang.String getDate()
  ```

  - **Description**

    Gets the date.

  - **Returns** – the date

- **getEntity**

  ```
  public java.lang.String getEntity()
  ```

  - **Description**

    Gets the entity.

  - **Returns** – the entity

- **getMetric**

  ```
  public java.lang.String getMetric()
  ```

  - **Description**

    Gets the metric.

  - **Returns** – the metric

- **getMonth**

  ```
  public java.lang.String getMonth()
  ```

- **Description**

  Gets the month.

- **Returns** – the month

- **getSection**

  ```
  public java.lang.String getSection()
  ```

  - **Description**

    Gets the section.

  - **Returns** – the section

- **getStatus**

  ```
  public java.lang.String getStatus()
  ```

  - **Description**

    Gets the status.

  - **Returns** – the status

- **getSubsection**

  ```
  public java.lang.String getSubsection()
  ```

  - **Description**

    Gets the subsection.

  - **Returns** – the subsection

- **setCertificate**

  ```
  public void setCertificate(java.lang.String certificate)
  ```

- **Description**

  Sets the certificate.

- **Parameters**

  * `certificate` – the new certificate

- **setCertificateDescription**

  ```
  public void setCertificateDescription(java.lang.String
      certificateDescription)
  ```

  - **Description**

    Sets the certificate description.

  - **Parameters**

    * `certificateDescription` – the new certificate description

- **setDate**

  ```
  public void setDate(java.lang.String date)
  ```

  - **Description**

    Sets the date.

  - **Parameters**

    * `date` – the new date

- **setEntity**

  ```
  public void setEntity(java.lang.String entity)
  ```

  - **Description**

    Sets the entity.

  - **Parameters**

* `entity` – the new entity

- **setIdMetrica**

  ```
  public void setIdMetrica(java.lang.String metric)
  ```

  - **Description**

    Sets the id metrica.

  - **Parameters**

    * `metric` – the new id metrica

- **setMonth**

  ```
  public void setMonth(java.lang.String month)
  ```

  - **Description**

    Sets the month.

  - **Parameters**

    * `month` – the new month

- **setSection**

  ```
  public void setSection(java.lang.String section)
  ```

  - **Description**

    Sets the section.

  - **Parameters**

    * `section` – the new section

- **setStatus**

  ```
  public void setStatus(java.lang.String status)
  ```

- **Description**

  Sets the status.

- **Parameters**

  * `status` – the new status

- **setSubsection**

```
public void setSubsection(java.lang.String subsection)
```

  - **Description**

    Sets the subsection.

  - **Parameters**

    * `subsection` – the new subsection

1.5 Class CertificateCondition

Declaration

```
public class CertificateCondition
 extends java.lang.Object
```

Field summary

boolErrorCondition
boolValidate
condition
errorCode
errorDescription
field
table

Constructor summary

CertificateCondition()

Method summary

getBoolErrorCondition()
getBoolValidate()
getCondition()
getErrorCode()
getErrorDescription()
getField()
getTable()
setBoolErrorCondition(String)
setBoolValidate(String)
setCondition(String)
setErrorCode(String)
setErrorDescription(String)
setField(String)
setTable(String)

Fields

- `private java.lang.String` **errorCode**
- `private java.lang.String` **errorDescription**
- `private java.lang.String` **condition**
- `private java.lang.String` **table**
- `private java.lang.String` **field**
- `private java.lang.String` **boolValidate**
- `private java.lang.String` **boolErrorCondition**

Constructors

- **CertificateCondition**

  public CertificateCondition()

Methods

- **getBoolErrorCondition**

  ```
  public java.lang.String getBoolErrorCondition()
  ```

- **getBoolValidate**

  ```
  public java.lang.String getBoolValidate()
  ```

- **getCondition**

  ```
  public java.lang.String getCondition()
  ```

- **getErrorCode**

  ```
  public java.lang.String getErrorCode()
  ```

- **getErrorDescription**

  ```
  public java.lang.String getErrorDescription()
  ```

- **getField**

  ```
  public java.lang.String getField()
  ```

- **getTable**

  ```
  public java.lang.String getTable()
  ```

- **setBoolErrorCondition**

  ```
  public void setBoolErrorCondition(java.lang.String
      boolErrorCondition)
  ```

- **setBoolValidate**

  ```
  public void setBoolValidate(java.lang.String boolValidate)
  ```

- **setCondition**

```
public void setCondition(java.lang.String condition)
```

- **setErrorCode**

```
public void setErrorCode(java.lang.String errorCode)
```

- **setErrorDescription**

```
public void setErrorDescription(java.lang.String
    errorDescription)
```

- **setField**

```
public void setField(java.lang.String field)
```

- **setTable**

```
public void setTable(java.lang.String table)
```

1.6 Class ComponentDescription

The Class ComponentDescription. Represents the fields that any component description of a dictionary element should have.

Declaration

```
public abstract class ComponentDescription
 extends java.lang.Object
```

All known subclasses

MasterAttributeDescription (in 1.14, page 183), IndicatorDescription (in 1.12, page 169), AttributeDescription (in 1.1, page 107)

Field summary

comments The comments.
definition The definition.
history The history.
id The id.
name The name.
obtentionMethod The obtention method.
responsible The responsible.

Constructor summary

ComponentDescription()

Method summary

getComments() Gets the comments.
getDefinition() Gets the definition.
getHistory() Gets the history.
getId() Gets the id.
getName() Gets the name.
getObtentionMethod() Gets the obtention method.
getResponsible() Gets the responsible.
setComments(String) Sets the comments.
setDefinition(String) Sets the definition.
setHistory(String) Sets the history.
setId(int) Sets the id.
setName(String) Sets the name.
setObtentionMethod(String) Sets the obtention method.
setResponsible(String) Sets the responsible.

Fields

- `protected int` **id**
  - The id.
- `protected java.lang.String` **name**
  - The name.

- `protected java.lang.String` **responsible**
  - The responsible.
- `protected java.lang.String` **definition**
  - The definition.
- `protected java.lang.String` **comments**
  - The comments.
- `protected java.lang.String` **history**
  - The history.
- `protected java.lang.String` **obtentionMethod**
  - The obtention method.

Constructors

- **ComponentDescription**

  ```
  public ComponentDescription()
  ```

Methods

- **getComments**

  ```
  public java.lang.String getComments()
  ```

  - **Description**

    Gets the comments.

  - **Returns** – the comments

- **getDefinition**

  ```
  public java.lang.String getDefinition()
  ```

- **Description**

  Gets the definition.

- **Returns** – the definition

- **getHistory**

  ```
  public java.lang.String getHistory()
  ```

  - **Description**

    Gets the history.

  - **Returns** – the history

- **getId**

  ```
  public int getId()
  ```

  - **Description**

    Gets the id.

  - **Returns** – the id

- **getName**

  ```
  public java.lang.String getName()
  ```

  - **Description**

    Gets the name.

  - **Returns** – the name

- **getObtentionMethod**

  ```
  public java.lang.String getObtentionMethod()
  ```

- **Description**

  Gets the obtention method.

- **Returns** – the obtention method

- **getResponsible**

  ```
  public java.lang.String getResponsible()
  ```

  - **Description**

    Gets the responsible.

  - **Returns** – the responsible

- **setComments**

  ```
  public void setComments(java.lang.String comments)
  ```

  - **Description**

    Sets the comments.

  - **Parameters**

    * `comments` – the new comments

- **setDefinition**

  ```
  public void setDefinition(java.lang.String definition)
  ```

  - **Description**

    Sets the definition.

  - **Parameters**

    * `definition` – the new definition

- **setHistory**

```
public void setHistory(java.lang.String history)
```

- **Description**

  Sets the history.

- **Parameters**

  * `history` – the new history

- **setId**

```
public void setId(int id)
```

- **Description**

  Sets the id.

- **Parameters**

  * `id` – the new id

- **setName**

```
public void setName(java.lang.String name)
```

- **Description**

  Sets the name.

- **Parameters**

  * `name` – the new name

- **setObtentionMethod**

```
public void setObtentionMethod(java.lang.String obtentionMethod)
```

- **Description**

  Sets the obtention method.

- **Parameters**
  - `obtentionMethod` – the new obtention method

- **setResponsible**

```
public void setResponsible(java.lang.String responsible)
```

  - **Description**

    Sets the responsible.

  - **Parameters**
    - `responsible` – the new responsible

1.7 Class ConceptTypeEnum

The Enum ConceptTypeEnum. Holds the possible types of the concepts in the dictionary of concepts.

Declaration

```
public final class ConceptTypeEnum
 extends java.lang.Enum
```

Field summary

ATRIBUTO The attribute.
ATRIBUTO_MAESTRO The master attribute.
DIMENSION The dimension.
ENTIDAD The entity.
INDICADOR The indicator.
JERARQUIA The hierarchy.
LITERAL The literal.
SECCION The section.
SUBSECCION The subsection.
UNKNOWN For safety, unknown if not any of the above.

Constructor summary

ConceptTypeEnum()

Method summary

valueOf(String)
values()

Fields

- `public static final ConceptTypeEnum` **SECCION**
  - The section.
- `public static final ConceptTypeEnum` **SUBSECCION**
  - The subsection.
- `public static final ConceptTypeEnum` **ENTIDAD**
  - The entity.
- `public static final ConceptTypeEnum` **DIMENSION**
  - The dimension.
- `public static final ConceptTypeEnum` **JERARQUIA**
  - The hierarchy.
- `public static final ConceptTypeEnum` **ATRIBUTO_MAESTRO**
  - The master attribute.
- `public static final ConceptTypeEnum` **ATRIBUTO**
  - The attribute.
- `public static final ConceptTypeEnum` **INDICADOR**
  - The indicator.
- `public static final ConceptTypeEnum` **LITERAL**

  - The literal.

- `public static final ConceptTypeEnum` **UNKNOWN**
  - For safety, unknown if not any of the above.

Constructors

- **ConceptTypeEnum**

  ```
  private ConceptTypeEnum()
  ```

Methods

- **valueOf**

  ```
  public static ConceptTypeEnum valueOf(java.lang.String name)
  ```

- **values**

  ```
  public static ConceptTypeEnum[] values()
  ```

Members inherited from class Enum

`java.lang.Enum`

- `protected final Object` **clone**`() throws CloneNotSupportedException`
- `public final int` **compareTo**`(Enum` **arg0**`)`
- `public final boolean` **equals**`(Object` **arg0**`)`
- `protected final void` **finalize**`()`
- `public final Class` **getDeclaringClass**`()`
- `public final int` **hashCode**`()`
- `private final` **name**
- `public final String` **name**`()`
- `private final` **ordinal**

- `public final int` **ordinal**`()`
- `private void` **readObject**`(java.io.ObjectInputStream` **arg0**`) throws java.io.IOException, java.lang.ClassNotFoundException`
- `private void` **readObjectNoData**`() throws java.io.ObjectStreamException`
- `public String` **toString**`()`
- `public static Enum` **valueOf**`(Class` **arg0**`, String` **arg1**`)`

1.8 Class Dependency

The Class Dependency. Represents a dependency among two jobs.

Declaration

```
public class Dependency
 extends java.lang.Object
```

Field summary

executionID The execution id.
fatherJobID The father job id.
flowID The flow id.
habilitattedSN The habilitatted sn.
software The software.
sonJobID The son job id.
status The status.
system The system.

Constructor summary

Dependency()

Method summary

getFatherJobID() Gets the father job id.
getFlowID() Gets the flow id.
getHabilitattedSN() Gets the if it is habilitatted.

getIdEjecucion() Gets the id ejecucion.
getSoftware() Gets the software.
getSonJobID() Gets the son job id.
getStatus() Gets the status.
getSystem() Gets the system.
setExecutionID(int) Sets the execution id.
setFatherJobID(String) Sets the father job id.
setFlowID(int) Sets the flow id.
setHabilitattedSN(String) Sets the habilitatted state.
setSoftware(int) Sets the software.
setSonJobID(String) Sets the son job id.
setStatus(String) Sets the status.
setSystem(String) Sets the system.

Fields

- `private int` **executionID**
  - The execution id.
- `private java.lang.String` **system**
  - The system.
- `private int` **software**
  - The software.
- `private int` **flowID**
  - The flow id.
- `private java.lang.String` **fatherJobID**
  - The father job id.
- `private java.lang.String` **sonJobID**
  - The son job id.
- `private java.lang.String` **habilitattedSN**
  - The habilitatted sn.
- `private StatusEnum` **status**

– The status.

Constructors

- **Dependency**

  ```
  public Dependency()
  ```

Methods

- **getFatherJobID**

  ```
  public java.lang.String getFatherJobID()
  ```

  – **Description**

  Gets the father job id.

  – **Returns** – the father job id

- **getFlowID**

  ```
  public int getFlowID()
  ```

  – **Description**

  Gets the flow id.

  – **Returns** – the flow id

- **getHabilitattedSN**

  ```
  public java.lang.String getHabilitattedSN()
  ```

  – **Description**

  Gets the if it is habilitatted.

- **Returns** – the habilitatted sn

- **getIdEjecucion**

  ```
  public int getIdEjecucion()
  ```

  - **Description**

    Gets the id ejecucion.

  - **Returns** – the id ejecucion

- **getSoftware**

  ```
  public int getSoftware()
  ```

  - **Description**

    Gets the software.

  - **Returns** – the software

- **getSonJobID**

  ```
  public java.lang.String getSonJobID()
  ```

  - **Description**

    Gets the son job id.

  - **Returns** – the son job id

- **getStatus**

  ```
  public java.lang.String getStatus()
  ```

  - **Description**

    Gets the status.

- **Returns** – the status

- **getSystem**

  ```
  public java.lang.String getSystem()
  ```

  - **Description**

    Gets the system.

  - **Returns** – the system

- **setExecutionID**

  ```
  public void setExecutionID(int executionID)
  ```

  - **Description**

    Sets the execution id.

  - **Parameters**

    * `executionID` – the new execution id

- **setFatherJobID**

  ```
  public void setFatherJobID(java.lang.String fatherJobID)
  ```

  - **Description**

    Sets the father job id.

  - **Parameters**

    * `fatherJobID` – the new father job id

- **setFlowID**

  ```
  public void setFlowID(int flowID)
  ```

- **Description**

  Sets the flow id.

- **Parameters**

  * `flowID` – the new flow id

- **setHabilitattedSN**

```
public void setHabilitattedSN(java.lang.String habilitattedSN)
```

- **Description**

  Sets the habilitatted state.

- **Parameters**

  * `habilitattedSN` – the new habilitatted sn

- **setSoftware**

```
public void setSoftware(int software)
```

- **Description**

  Sets the software.

- **Parameters**

  * `software` – the new software

- **setSonJobID**

```
public void setSonJobID(java.lang.String sonJobID)
```

- **Description**

  Sets the son job id.

- **Parameters**

* `sonJobID` – the new son job id

- **setStatus**

  ```
  public void setStatus(java.lang.String status)
  ```

  – **Description**

  Sets the status.

  – **Parameters**

  * `status` – the new status

- **setSystem**

  ```
  public void setSystem(java.lang.String system)
  ```

  – **Description**

  Sets the system.

  – **Parameters**

  * `system` – the new system

1.9 Class DictionaryConcept

The Class DictionaryConcept. Represents a concept of the dictionary.

Declaration

```
public class DictionaryConcept
 extends java.lang.Object
```

Field summary

compRowID The Oracle rowID of the component.

concept The concept.
ctRowID The Oracle rowID of the ct(?).
description The description of the concept, if it is an attribute or an indicator.
level The level.
status The status.
type The type.

Constructor summary

DictionaryConcept() Instantiates a new dictionary concept.
DictionaryConcept(String) Instantiates a new dictionary concept.
DictionaryConcept(String, int) Instantiates a new dictionary concept.
DictionaryConcept(String, int, int, ConceptTypeEnum) Instantiates a new dictionary concept.
DictionaryConcept(String, int, int, ConceptTypeEnum, String, String) Instantiates a new dictionary concept.

Method summary

getCompRowID() Gets the Oracle rowID of the component
getConcept() Gets the concept.
getCtRowID() Gets the Oracle rowID of the ct(?)
getLevel() Gets the level.
getStatus() Gets the status.
getType() Gets the type.
setCompRowID(String) Sets the Oracle rowID of the component
setConcept(String) Sets the concept.
setCtRowID(String) Sets the Oracle rowID of the ct(?)
setLevel(int) Sets the level.
setStatus(int) Sets the status.
setType(ConceptTypeEnum) Sets the type.
toString()

Fields

- `private java.lang.String` **concept**
  - The concept.
- `private int` **level**

  - The level.

- `private int` **status**
  - The status.
- `private ConceptTypeEnum` **type**
  - The type.
- `private java.lang.String` **compRowID**
  - The Oracle rowID of the component.
- `private java.lang.String` **ctRowID**
  - The Oracle rowID of the ct(?).
- `private ComponentDescription` **description**
  - The description of the concept, if it is an attribute or an indicator.

Constructors

- **DictionaryConcept**

  ```
  public DictionaryConcept()
  ```

  - **Description**

    Instantiates a new dictionary concept.

- **DictionaryConcept**

  ```
  public DictionaryConcept(java.lang.String concept)
  ```

  - **Description**

    Instantiates a new dictionary concept.

  - **Parameters**

    * `concept` – the concept

- **DictionaryConcept**

  public DictionaryConcept(java.lang.String concept, **int** level)

  - **Description**

    Instantiates a new dictionary concept.

  - **Parameters**

    * `concept` – the concept

    * `level` – the level

- **DictionaryConcept**

  public DictionaryConcept(java.lang.String concept, **int** level, **int** status, ConceptTypeEnum tipo)

  - **Description**

    Instantiates a new dictionary concept.

  - **Parameters**

    * `concept` – the concept

    * `level` – the level

    * `status` – the status

    * `tipo` – the tipo

- **DictionaryConcept**

  public DictionaryConcept(java.lang.String concept, **int** level, **int** status, ConceptTypeEnum tipo, java.lang.String compRowID, java.lang.String ctRowID)

  - **Description**

    Instantiates a new dictionary concept.

- **Parameters**
  - `concept` – the concept
  - `level` – the level
  - `status` – the status
  - `tipo` – the tipo
  - `compRowID` – the comp row id
  - `ctRowID` – the ct row id

Methods

- **getCompRowID**

  `public java.lang.String getCompRowID()`

  - **Description**

    Gets the Oracle rowID of the component

  - **Returns** – the rowID

- **getConcept**

  `public java.lang.String getConcept()`

  - **Description**

    Gets the concept.

  - **Returns** – the concept

- **getCtRowID**

  `public java.lang.String getCtRowID()`

- **Description**

  Gets the Oracle rowID of the ct(?)

- **Returns** – the rowID

- **getLevel**

  ```
  public int getLevel()
  ```

  - **Description**

    Gets the level.

  - **Returns** – the level

- **getStatus**

  ```
  public int getStatus()
  ```

  - **Description**

    Gets the status.

  - **Returns** – the status

- **getType**

  ```
  public ConceptTypeEnum getType()
  ```

  - **Description**

    Gets the type.

  - **Returns** – the type

- **setCompRowID**

  ```
  public void setCompRowID(java.lang.String compRowID)
  ```

- **Description**

  Sets the Oracle rowID of the component

- **Parameters**

  * `compRowID` – the new rowID

- **setConcept**

  ```
  public void setConcept(java.lang.String concept)
  ```

  - **Description**

    Sets the concept.

  - **Parameters**

    * `concept` – the new concept

- **setCtRowID**

  ```
  public void setCtRowID(java.lang.String ctRowID)
  ```

  - **Description**

    Sets the Oracle rowID of the ct(?)

  - **Parameters**

    * `ctRowID` – the new rowID

- **setLevel**

  ```
  public void setLevel(int level)
  ```

  - **Description**

    Sets the level.

  - **Parameters**

* `level` – the new level

- **setStatus**

```
public void setStatus(int status)
```

  - **Description**

    Sets the status.

  - **Parameters**

    * `status` – the new status

- **setType**

```
public void setType(ConceptTypeEnum type)
```

  - **Description**

    Sets the type.

  - **Parameters**

    * `type` – the new type

- **toString**

```
public java.lang.String toString()
```

1.10 Class Execution

The Class Execution. Represent an instance of an ETL `Flow` (in 1.11, page 162) for a given date.

Declaration

```
public class Execution
 extends java.lang.Object
```

Field summary

dataDate The date of data.
duration The duration.
endDate The end date.
flow The flow.
id The id.
oFlow The proper object flow of which the execution is an instance.
plannedDate The planned date.
scenario The scenario.
software The software.
startDate The start date.
status The status.

Constructor summary

Execution()

Method summary

getDataDate() Gets the date of data.
getDuration() Gets the duration.
getEndDate() Gets the end date.
getFlow() Gets the flow.
getId() Gets the id.
getPlannedDate() Gets the planned date.
getScenario() Gets the scenario.
getSoftware() Gets the software.
getStartDate() Gets the start date.
getStatus() Gets the status.
setDataDate(String) Sets the date of data.
setDuration(String) Sets the duration.
setEndDate(Date) Sets the end date.
setFlow(String) Sets the flow.
setId(int) Sets the id.
setPlannedDate(Date) Sets the planned date.
setScenario(String) Sets the scenario.
setSoftware(String) Sets the software.
setStartDate(Date) Sets the start date.
setStatus(String) Sets the status.
toString()

Fields

- `private int` **id**
  - The id.
- `private java.lang.String` **flow**
  - The flow.
- `private Flow` **oFlow**
  - The proper object flow of which the execution is an instance.
- `private StatusEnum` **status**
  - The status.
- `private java.lang.String` **dataDate**
  - The date of data.
- `private java.lang.String` **scenario**
  - The scenario.
- `private java.sql.Date` **startDate**
  - The start date.
- `private java.sql.Date` **endDate**
  - The end date.
- `private java.sql.Date` **plannedDate**
  - The planned date.
- `private java.lang.String` **software**
  - The software.
- `private java.lang.String` **duration**
  - The duration.

Constructors

- **Execution**

  ```
  public Execution()
  ```

Methods

- **getDataDate**

  ```
  public java.lang.String getDataDate()
  ```

  - **Description**

    Gets the date of data.

  - **Returns** – the date of data

- **getDuration**

  ```
  public java.lang.String getDuration()
  ```

  - **Description**

    Gets the duration.

  - **Returns** – the duration

- **getEndDate**

  ```
  public java.sql.Date getEndDate()
  ```

  - **Description**

    Gets the end date.

  - **Returns** – the end date

- **getFlow**

  ```
  public java.lang.String getFlow()
  ```

  - **Description**

    Gets the flow.

  - **Returns** – the flow

- **getId**

  ```
  public int getId()
  ```

  - **Description**

    Gets the id.

  - **Returns** – the id

- **getPlannedDate**

  ```
  public java.sql.Date getPlannedDate()
  ```

  - **Description**

    Gets the planned date.

  - **Returns** – the planned date

- **getScenario**

  ```
  public java.lang.String getScenario()
  ```

  - **Description**

    Gets the scenario.

  - **Returns** – the scenario

- **getSoftware**

  ```
  public java.lang.String getSoftware()
  ```

  - **Description**

    Gets the software.

  - **Returns** – the software

- **getStartDate**

  ```
  public java.sql.Date getStartDate()
  ```

  - **Description**

    Gets the start date.

  - **Returns** – the start date

- **getStatus**

  ```
  public java.lang.String getStatus()
  ```

  - **Description**

    Gets the status.

  - **Returns** – the status

- **setDataDate**

  ```
  public void setDataDate(java.lang.String dataDate)
  ```

  - **Description**

    Sets the date of data.

  - **Parameters**

    * `dataDate` – the new date of data

- **setDuration**

  ```
  public void setDuration(java.lang.String duration)
  ```

  - **Description**

    Sets the duration.

  - **Parameters**

    * `duration` – the new duration

- **setEndDate**

  ```
  public void setEndDate(java.sql.Date endDate)
  ```

  - **Description**

    Sets the end date.

  - **Parameters**

    * `endDate` – the new end date

- **setFlow**

  ```
  public void setFlow(java.lang.String flow)
  ```

  - **Description**

    Sets the flow.

  - **Parameters**

    * `flow` – the new flow

- **setId**

  ```
  public void setId(int id)
  ```

- **Description**

  Sets the id.

- **Parameters**

  * `id` – the new id

- **setPlannedDate**

  ```
  public void setPlannedDate(java.sql.Date plannedDate)
  ```

  - **Description**

    Sets the planned date.

  - **Parameters**

    * `plannedDate` – the new planned date

- **setScenario**

  ```
  public void setScenario(java.lang.String scenario)
  ```

  - **Description**

    Sets the scenario.

  - **Parameters**

    * `scenario` – the new scenario

- **setSoftware**

  ```
  public void setSoftware(java.lang.String software)
  ```

  - **Description**

    Sets the software.

  - **Parameters**

* `software` – the new software

- **setStartDate**

  ```
  public void setStartDate(java.sql.Date startDate)
  ```

  – **Description**

  Sets the start date.

  – **Parameters**

  * `startDate` – the new start date

- **setStatus**

  ```
  public void setStatus(java.lang.String status)
  ```

  – **Description**

  Sets the status.

  – **Parameters**

  * `status` – the new status

- **toString**

  ```
  public java.lang.String toString()
  ```

1.11 Class Flow

The Class Flow. Represents an ETL flow for a system and software version. A flow can't (shouldn't) get any of its fields modified after it's been executed as an `Execution` (in 1.10, page 154).

Declaration

```
public class Flow
 extends java.lang.Object
```

Field summary

dependencies The dependencies among the jobs of this flow.
id The unique immutable identifier.
isAtypical Indicates if the flow is typical or atypical.
jobs The identifiers of the jobs that will run on this flow.
name The descriptive name of the flow.
oJobs The proper object jobs of the flow.
software The software version of the system.
system The system on which the flow will be executed.

Constructor summary

Flow() Instantiates a new empty flow.
Flow(String, String, String[], Map) Instantiates a new flow with a given name, typicality, list of jobs and list of dependencies.

Method summary

getDependencies() Gets the dependencies of the jobs of associated to this flow.
getId() Gets the unique identifier.
getIsAtypical() Gets the typicality of the flow.
getJobs() Gets the identifiers of the jobs of this flow.
getName() Gets the name of the flow.
getSoftware() Gets the software version of the system.
getSystem() Gets the system where the flow is defined.
setDependencies(Map) Sets the dependencies of the jobs associated to this flow.
setId(int) Sets the unique identifier.
setIsAtypical(String) Sets the typicality of the flow.
setJobs(String[]) Sets the identifiers of the jobs of this flow.
setName(String) Sets the name of the flow.
setSoftware(String) Sets the software version for a new flow.
setSystem(String) Sets the system on which the flow will run.

Fields

- `private int` **id**
  - The unique immutable identifier.
- `private java.lang.String` **name**
  - The descriptive name of the flow.
- `private java.lang.String` **isAtypical**
  - Indicates if the flow is typical or atypical. Can me either 'S' or 'N', corresponding to typical and atypical.
- `private java.lang.String` **system**
  - The system on which the flow will be executed.
- `private java.lang.String` **software**
  - The software version of the system.
- `private java.lang.String[]` **jobs**
  - The identifiers of the jobs that will run on this flow. See `Job` (in 1.13, page 173)
- `private java.util.List` **oJobs**
  - The proper object jobs of the flow.
- `private java.util.Map` **dependencies**
  - The dependencies among the jobs of this flow. The key is the identifier of a job j, the values are the identifiers of the jobs that must finish successfully before j can be executed. See `Dependency` (in 1.8, page 141)

Constructors

- **Flow**

  `public Flow()`

  - **Description**

Instantiates a new empty flow.

- **Flow**

```
public Flow(java.lang.String name,java.lang.String isAtypical,
    java.lang.String[] jobs,java.util.Map dependencies)
```

  - **Description**

    Instantiates a new flow with a given name, typicality, list of jobs and list of dependencies. System and software will be set to those active in the current environment. See `Environment` (in 4.1, page 263) .

  - **Parameters**

    * `name` – the name of the flow
    * `isAtypical` – the typicality
    * `jobs` – the list of jobs
    * `dependencies` – the dependencies of the jobs

Methods

- **getDependencies**

```
public java.util.Map getDependencies()
```

  - **Description**

    Gets the dependencies of the jobs of associated to this flow.

  - **Returns** – a map with the dependencies

- **getId**

```
public int getId()
```

- **Description**

  Gets the unique identifier.

- **Returns** – the unique identifier

- **getIsAtypical**

```
public java.lang.String getIsAtypical()
```

  - **Description**

    Gets the typicality of the flow.

  - **Returns** – the typicality

- **getJobs**

```
public java.lang.String[] getJobs()
```

  - **Description**

    Gets the identifiers of the jobs of this flow.

  - **Returns** – the identifiers of the jobs

- **getName**

```
public java.lang.String getName()
```

  - **Description**

    Gets the name of the flow.

  - **Returns** – the name of the flow

- **getSoftware**

```
public java.lang.String getSoftware()
```

- **Description**

  Gets the software version of the system.

- **Returns** – the software version

- **getSystem**

```
public java.lang.String getSystem()
```

  - **Description**

    Gets the system where the flow is defined.

  - **Returns** – the system

- **setDependencies**

```
public void setDependencies(java.util.Map dependencies)
```

  - **Description**

    Sets the dependencies of the jobs associated to this flow.

  - **Parameters**

    * `dependencies` – a map whose key is the identifier of a job and the value is an array with the identifiers of the jobs on which that depends

- **setId**

```
public void setId(int id)
```

  - **Description**

    Sets the unique identifier. Can't (shouldn't) be set more than once.

  - **Parameters**

    * `id` – the new unique identifier

- **setIsAtypical**

  ```
  public void setIsAtypical(java.lang.String isAtypical)
  ```

  - **Description**

    Sets the typicality of the flow.

  - **Parameters**

    * `isAtypical` – the new es atipico

- **setJobs**

  ```
  public void setJobs(java.lang.String[] jobs)
  ```

  - **Description**

    Sets the identifiers of the jobs of this flow.

  - **Parameters**

    * `jobs` – an array containing the identifiers of the jobs

- **setName**

  ```
  public void setName(java.lang.String name)
  ```

  - **Description**

    Sets the name of the flow.

  - **Parameters**

    * `name` – the new name

- **setSoftware**

  ```
  public void setSoftware(java.lang.String software)
  ```

- **Description**

  Sets the software version for a new flow.

- **Parameters**

  * `software` – the software version

- **setSystem**

```
public void setSystem(java.lang.String system)
```

  - **Description**

    Sets the system on which the flow will run.

  - **Parameters**

    * `system` – the system

1.12 Class IndicatorDescription

The Class IndicatorDescription. Represents the description of an indicator element in the dictionary of concepts.

Declaration

```
public class IndicatorDescription
 extends com.indra.iquality.model.ComponentDescription
```

Field summary

acummulatedPeriod The acummulated period.
certificates The certificates.
measureUnits The measure units.

Constructor summary

IndicatorDescription()

Method summary

getAcummulatedPeriod() Gets the acummulated period.
getCertificates() Gets the certificates.
getMeasureUnit() Gets the measure unit.
setAcummulatedPeriod(String) Sets the acummulated period.
setCertificates(List) Sets the certificates.
setMeasureUnit(String) Sets the measure unit.

Fields

- `private java.lang.String` **measureUnits**
  - The measure units.
- `private java.lang.String` **acummulatedPeriod**
  - The acummulated period.
- `private java.util.List` **certificates**
  - The certificates.

Constructors

- **IndicatorDescription**

  ```
  public IndicatorDescription()
  ```

Methods

- **getAcummulatedPeriod**

  ```
  public java.lang.String getAcummulatedPeriod()
  ```

  - **Description**

    Gets the acummulated period.

- **Returns** – the acummulated period

- **getCertificates**

  ```
  public java.util.List getCertificates()
  ```

  - **Description**

    Gets the certificates.

  - **Returns** – the certificates

- **getMeasureUnit**

  ```
  public java.lang.String getMeasureUnit()
  ```

  - **Description**

    Gets the measure unit.

  - **Returns** – the measure unit

- **setAcummulatedPeriod**

  ```
  public void setAcummulatedPeriod(java.lang.String
      acummulatedPeriod)
  ```

  - **Description**

    Sets the acummulated period.

  - **Parameters**

    * `acummulatedPeriod` – the new acummulated period

- **setCertificates**

  ```
  public void setCertificates(java.util.List certificates)
  ```

- **Description**

  Sets the certificates.

- **Parameters**

  * `certificates` – the new certificates

- **setMeasureUnit**

  ```
  public void setMeasureUnit(java.lang.String measureUnits)
  ```

  - **Description**

    Sets the measure unit.

  - **Parameters**

    * `measureUnits` – the new measure unit

Members inherited from class ComponentDescription

`com.indra.iquality.model.ComponentDescription` (in 1.6, page 132)

- `protected` **comments**
- `protected` **definition**
- `public String` **getComments()**
- `public String` **getDefinition()**
- `public String` **getHistory()**
- `public int` **getId()**
- `public String` **getName()**
- `public String` **getObtentionMethod()**
- `public String` **getResponsible()**
- `protected` **history**
- `protected` **id**
- `protected` **name**
- `protected` **obtentionMethod**

- `protected` **responsible**
- `public void` **setComments**`(java.lang.String` **comments**`)`
- `public void` **setDefinition**`(java.lang.String` **definition**`)`
- `public void` **setHistory**`(java.lang.String` **history**`)`
- `public void` **setId**`(int` **id**`)`
- `public void` **setName**`(java.lang.String` **name**`)`
- `public void` **setObtentionMethod**`(java.lang.String` **obtentionMethod**`)`
- `public void` **setResponsible**`(java.lang.String` **responsible**`)`

1.13 Class Job

The Class Job. Represents an atomic operation of an ETL `Execution` (in 1.10, page 154).

Declaration

```
public class Job
 extends java.lang.Object
```

Field summary

block The block.
blockID The block id.
checkpoint The checkpoint.
checkpointLastOKDate The checkpoint last ok date.
dependencies The dependencies of the job.
duration The duration.
endDate The end date.
executionID The execution id.
jobID The job id.
regops The register of operations of the job.
software The software.
startDate The start date.
status The status.
system The system.

Constructor summary

Job()

Method summary

getBlock() Gets the block.
getBlockID() Gets the block id.
getCheckpoint() Gets the checkpoint.
getCheckpointLastOKDate() Gets the checkpoint last ok date.
getDuration() Gets the duration.
getEndDate() Gets the end date.
getExecutionID() Gets the execution id.
getJobID() Gets the job id.
getSistema() Gets the sistema.
getSoftware() Gets the software.
getStartDate() Gets the start date.
getStatus() Gets the status.
setBlock(String) Sets the block.
setBlockID(String) Sets the block id.
setCheckpoint(String) Sets the checkpoint.
setCheckpointLastOKDate(Date) Sets the checkpoint last ok date.
setDuration(String) Sets the duration.
setEndDate(Date) Sets the end date.
setExecutionID(String) Sets the execution id.
setJobID(String) Sets the job id.
setSoftware(int) Sets the software.
setStartDate(Date) Sets the start date.
setStatus(String) Sets the status.
setSystem(String) Sets the system.
toString() To string.

Fields

- `private java.lang.String` **jobID**
  - The job id.
- `private java.lang.String` **executionID**
  - The execution id.

- `private StatusEnum` **status**
  - The status.
- `private java.sql.Date` **startDate**
  - The start date.
- `private java.sql.Date` **endDate**
  - The end date.
- `private java.lang.String` **checkpoint**
  - The checkpoint.
- `private java.sql.Date` **checkpointLastOKDate**
  - The checkpoint last ok date.
- `private java.lang.String` **duration**
  - The duration.
- `private java.lang.String` **blockID**
  - The block id.
- `private java.lang.String` **block**
  - The block.
- `private java.lang.String` **system**
  - The system.
- `private int` **software**
  - The software.
- `private java.util.List` **regops**
  - The register of operations of the job.
- `private java.util.List` **dependencies**
  - The dependencies of the job.

Constructors

- **Job**

  ```
  public Job()
  ```

Methods

- **getBlock**

  ```
  public java.lang.String getBlock()
  ```

  - **Description**

    Gets the block.

  - **Returns** – the block

- **getBlockID**

  ```
  public java.lang.String getBlockID()
  ```

  - **Description**

    Gets the block id.

  - **Returns** – the block id

- **getCheckpoint**

  ```
  public java.lang.String getCheckpoint()
  ```

  - **Description**

    Gets the checkpoint.

  - **Returns** – the checkpoint

- **getCheckpointLastOKDate**

  ```
  public java.sql.Date getCheckpointLastOKDate()
  ```

  - **Description**

    Gets the checkpoint last ok date.

  - **Returns** – the checkpoint last ok date

- **getDuration**

  ```
  public java.lang.String getDuration()
  ```

  - **Description**

    Gets the duration.

  - **Returns** – the duration

- **getEndDate**

  ```
  public java.sql.Date getEndDate()
  ```

  - **Description**

    Gets the end date.

  - **Returns** – the end date

- **getExecutionID**

  ```
  public java.lang.String getExecutionID()
  ```

  - **Description**

    Gets the execution id.

  - **Returns** – the execution id

- **getJobID**

  ```
  public java.lang.String getJobID()
  ```

  - **Description**

    Gets the job id.

  - **Returns** – the job id

- **getSistema**

  ```
  public java.lang.String getSistema()
  ```

  - **Description**

    Gets the sistema.

  - **Returns** – the sistema

- **getSoftware**

  ```
  public int getSoftware()
  ```

  - **Description**

    Gets the software.

  - **Returns** – the software

- **getStartDate**

  ```
  public java.sql.Date getStartDate()
  ```

  - **Description**

    Gets the start date.

  - **Returns** – the start date

- **getStatus**

  ```
  public java.lang.String getStatus()
  ```

  - **Description**

    Gets the status.

  - **Returns** – the status

- **setBlock**

  ```
  public void setBlock(java.lang.String block)
  ```

  - **Description**

    Sets the block.

  - **Parameters**

    * `block` – the new block

- **setBlockID**

  ```
  public void setBlockID(java.lang.String blockID)
  ```

  - **Description**

    Sets the block id.

  - **Parameters**

    * `blockID` – the new block id

- **setCheckpoint**

  ```
  public void setCheckpoint(java.lang.String checkpoint)
  ```

  - **Description**

    Sets the checkpoint.

- **Parameters**
  - `checkpoint` – the new checkpoint

- **setCheckpointLastOKDate**

```
public void setCheckpointLastOKDate(java.sql.Date
    checkpointLastOKDate)
```

  - **Description**

    Sets the checkpoint last ok date.

  - **Parameters**
    - `checkpointLastOKDate` – the new checkpoint last ok date

- **setDuration**

```
public void setDuration(java.lang.String duration)
```

  - **Description**

    Sets the duration.

  - **Parameters**
    - `duration` – the new duration

- **setEndDate**

```
public void setEndDate(java.sql.Date endDate)
```

  - **Description**

    Sets the end date.

  - **Parameters**
    - `endDate` – the new end date

- **setExecutionID**

  ```
  public void setExecutionID(java.lang.String executionID)
  ```

  - **Description**

    Sets the execution id.

  - **Parameters**

    * `executionID` – the new execution id

- **setJobID**

  ```
  public void setJobID(java.lang.String jobID)
  ```

  - **Description**

    Sets the job id.

  - **Parameters**

    * `jobID` – the new job id

- **setSoftware**

  ```
  public void setSoftware(int software)
  ```

  - **Description**

    Sets the software.

  - **Parameters**

    * `software` – the new software

- **setStartDate**

  ```
  public void setStartDate(java.sql.Date startDate)
  ```

- **Description**

  Sets the start date.

- **Parameters**

  * `startDate` – the new start date

- **setStatus**

  ```
  public void setStatus(java.lang.String status)
  ```

  - **Description**

    Sets the status.

  - **Parameters**

    * `status` – the new status

- **setSystem**

  ```
  public void setSystem(java.lang.String system)
  ```

  - **Description**

    Sets the system.

  - **Parameters**

    * `system` – the new system

- **toString**

  ```
  public java.lang.String toString()
  ```

  - **Description**

    To string.

  - **Returns** – the string

1.14 Class MasterAttributeDescription

The Class MasterAttributeDescription. Represents the description of an master attribute element in the dictionary of concepts.

Declaration

```
public class MasterAttributeDescription
 extends com.indra.iquality.model.AttributeDescription
```

Field summary

historyMaster The history master.
obtentionMethodMaster The obtention method master.
updatePeriodMaster The update period master.
updateTypeMaster The update type master.

Constructor summary

MasterAttributeDescription() Instantiates a new master attribute description.
MasterAttributeDescription(AttributeDescription) Instantiates a new master attribute description from an attribute description.

Method summary

getHistoryMaster() Gets the history master.
getObtentionMethodMaster() Gets the obtention method master.
getUpdatePeriodMaster() Gets the update period master.
getUpdateTypeMaster() Gets the update type master.
setHistoryMaster(String) Sets the history master.
setObtentionMethodMaster(String) Sets the obtention method master.
setUpdatePeriodMaster(String) Sets the update period master.
setUpdateTypeMaster(String) Sets the update type master.

Fields

- `private java.lang.String` **historyMaster**

  - The history master.

- `private java.lang.String` **updatePeriodMaster**
  - The update period master.
- `private java.lang.String` **updateTypeMaster**
  - The update type master.
- `private java.lang.String` **obtentionMethodMaster**
  - The obtention method master.

Constructors

- **MasterAttributeDescription**

  ```
  public MasterAttributeDescription()
  ```

  - **Description**

    Instantiates a new master attribute description.

- **MasterAttributeDescription**

  ```
  public MasterAttributeDescription(AttributeDescription da)
  ```

  - **Description**

    Instantiates a new master attribute description from an attribute description.

  - **Parameters**

    * `da` – the attribute description

Methods

- **getHistoryMaster**

```
public java.lang.String getHistoryMaster()
```

- **Description**

  Gets the history master.

- **Returns** – the history master

- **getObtentionMethodMaster**

```
public java.lang.String getObtentionMethodMaster()
```

- **Description**

  Gets the obtention method master.

- **Returns** – the obtention method master

- **getUpdatePeriodMaster**

```
public java.lang.String getUpdatePeriodMaster()
```

- **Description**

  Gets the update period master.

- **Returns** – the update period master

- **getUpdateTypeMaster**

```
public java.lang.String getUpdateTypeMaster()
```

- **Description**

  Gets the update type master.

- **Returns** – the update type master

- **setHistoryMaster**

public void setHistoryMaster(java.lang.String historyMaster)

- **Description**

  Sets the history master.

- **Parameters**

  * `historyMaster` – the new history master

- **setObtentionMethodMaster**

  public void setObtentionMethodMaster(java.lang.String obtentionMethodMaster)

  - **Description**

    Sets the obtention method master.

  - **Parameters**

    * `obtentionMethodMaster` – the new obtention method master

- **setUpdatePeriodMaster**

  public void setUpdatePeriodMaster(java.lang.String updatePeriodMaster)

  - **Description**

    Sets the update period master.

  - **Parameters**

    * `updatePeriodMaster` – the new update period master

- **setUpdateTypeMaster**

  public void setUpdateTypeMaster(java.lang.String updateTypeMaster)

– **Description**

Sets the update type master.

– **Parameters**

* `updateTypeMaster` – the new update type master

Members inherited from class AttributeDescription

`com.indra.iquality.model.AttributeDescription` (in 1.1, page 107)

- `protected` **format**
- `public String` **getFormat**`()`
- `public String` **getUpdatePeriod**`()`
- `public String` **getUpdateType**`()`
- `public void` **setFormat**`(java.lang.String` **format**`)`
- `public void` **setUodatePeriod**`(java.lang.String` **updatePeriod**`)`
- `public void` **setUpdateType**`(java.lang.String` **updateType**`)`
- `protected` **updatePeriod**
- `protected` **updateType**

Members inherited from class ComponentDescription

`com.indra.iquality.model.ComponentDescription` (in 1.6, page 132)

- `protected` **comments**
- `protected` **definition**
- `public String` **getComments**`()`
- `public String` **getDefinition**`()`
- `public String` **getHistory**`()`
- `public int` **getId**`()`
- `public String` **getName**`()`
- `public String` **getObtentionMethod**`()`
- `public String` **getResponsible**`()`
- `protected` **history**

- `protected` **id**
- `protected` **name**
- `protected` **obtentionMethod**
- `protected` **responsible**
- `public void` **setComments**`(java.lang.String` **comments**`)`
- `public void` **setDefinition**`(java.lang.String` **definition**`)`
- `public void` **setHistory**`(java.lang.String` **history**`)`
- `public void` **setId**`(int` **id**`)`
- `public void` **setName**`(java.lang.String` **name**`)`
- `public void` **setObtentionMethod**`(java.lang.String` **obtentionMethod**`)`
- `public void` **setResponsible**`(java.lang.String` **responsible**`)`

1.15 Class OperationTrace

The Class RegisterTrace. Represents the trace or log of a run job.

Declaration

```
public class OperationTrace
 extends java.lang.Object
```

Field summary

category The category.
date The date.
dateID The date id.
id The id.
message The message.

Constructor summary

OperationTrace()

Method summary

getCategory() Gets the category.
getDate() Gets the date.
getDateID() Gets the date id.
getId() Gets the id.
getMessage() Gets the message.
setCategory(String) Sets the category.
setDate(String) Sets the date.
setDateID(Date) Sets the date id.
setId(int) Sets the id.
setMessage(String) Sets the message.
toString()

Fields

- `private int` **id**
  - The id.
- `private java.sql.Date` **dateID**
  - The date id.
- `private java.lang.String` **date**
  - The date.
- `private java.lang.String` **category**
  - The category.
- `private java.lang.String` **message**
  - The message.

Constructors

- **OperationTrace**

```
public OperationTrace()
```

Methods

- **getCategory**

  ```
  public java.lang.String getCategory()
  ```

  - **Description**

    Gets the category.

  - **Returns** – the category

- **getDate**

  ```
  public java.lang.String getDate()
  ```

  - **Description**

    Gets the date.

  - **Returns** – the date

- **getDateID**

  ```
  public java.sql.Date getDateID()
  ```

  - **Description**

    Gets the date id.

  - **Returns** – the date id

- **getId**

  ```
  public int getId()
  ```

  - **Description**

    Gets the id.

- **Returns** – the id

- **getMessage**

  ```
  public java.lang.String getMessage()
  ```

  - **Description**

    Gets the message.

  - **Returns** – the message

- **setCategory**

  ```
  public void setCategory(java.lang.String category)
  ```

  - **Description**

    Sets the category.

  - **Parameters**

    * `category` – the new category

- **setDate**

  ```
  public void setDate(java.lang.String date)
  ```

  - **Description**

    Sets the date.

  - **Parameters**

    * `date` – the new date

- **setDateID**

  ```
  public void setDateID(java.sql.Date dateID)
  ```

- **Description**

  Sets the date id.

- **Parameters**

  * `dateID` – the new date id

- **setId**

  ```
  public void setId(int id)
  ```

  - **Description**

    Sets the id.

  - **Parameters**

    * `id` – the new id

- **setMessage**

  ```
  public void setMessage(java.lang.String message)
  ```

  - **Description**

    Sets the message.

  - **Parameters**

    * `message` – the new message

- **toString**

  ```
  public java.lang.String toString()
  ```

1.16 Class RegisterOfOperation

The Class RegisterOfOperation. Represents the register of operation of a `Job` (in 1.13, page 173).

Declaration

```
public class RegisterOfOperation
 extends java.lang.Object
```

Field summary

dataDate The data date.
durationFC The duration fc.
endDate The end date.
fcRowsDismissed The fc rows dismissed.
fcRowsLoaded The fc rows loaded.
fcRowsRead The fc rows read.
fcRowsRejected The fc rows rejected.
fcRowsUpdated The fc rows updated.
finalRowID The final row id.
operationID The operation id.
operationRowID The operation row id.
operationType The operation type.
opTrace The trace of actions done by the operation.
startDate The start date.
status The status.

Constructor summary

RegisterOfOperation()

Method summary

getDataDate() Gets the data date.
getDurationFC() Gets the duration fc.
getEndDate() Gets the end date.
getFcRowsDismissed() Gets the fc rows dismissed.
getFcRowsLoaded() Gets the fc rows loaded.
getFcRowsRead() Gets the fc rows read.
getFcRowsRejected() Gets the fc rows rejected.
getFcRowsUpdated() Gets the fc rows updated.
getFinalRowID() Gets the final row id.
getOperationID() Gets the operation id.
getOperationRowID() Gets the operation row id.
getOperationType() Gets the operation type.

getStartDate() Gets the start date.
getStatus() Gets the status.
setDataDate(Date) Sets the data date.
setDurationFC(double) Sets the duration fc.
setEndDate(Date) Sets the end date.
setFcRowsDismissed(int) Sets the fc rows dismissed.
setFcRowsLoaded(int) Sets the fc rows loaded.
setFcRowsRead(int) Sets the fc rows read.
setFcRowsRejected(int) Sets the fc rows rejected.
setFcRowsUpdated(int) Sets the fc rows updated.
setFinalRowID(String) Sets the final row id.
setOperationID(int) Sets the operation id.
setOperationRowID(String) Sets the operation row id.
setOperationType(String) Sets the operation type.
setStartDate(Date) Sets the start date.
setStatus(String) Sets the status.
toString()

Fields

- `private java.lang.String` **operationRowID**
  - The operation row id.
- `private java.lang.String` **finalRowID**
  - The final row id.
- `private int` **operationID**
  - The operation id.
- `private java.sql.Date` **startDate**
  - The start date.
- `private java.sql.Date` **endDate**
  - The end date.
- `private java.sql.Date` **dataDate**
  - The data date.
- `private double` **durationFC**

  - The duration fc.

- `private java.lang.String` **operationType**
  - The operation type.
- `private int` **fcRowsLoaded**
  - The fc rows loaded.
- `private int` **fcRowsUpdated**
  - The fc rows updated.
- `private int` **fcRowsRead**
  - The fc rows read.
- `private int` **fcRowsRejected**
  - The fc rows rejected.
- `private int` **fcRowsDismissed**
  - The fc rows dismissed.
- `private StatusEnum` **status**
  - The status.
- `private java.util.List` **opTrace**
  - The trace of actions done by the operation.

Constructors

- **RegisterOfOperation**

  public RegisterOfOperation()

Methods

- **getDataDate**

`public java.sql.Date getDataDate()`

- **Description**

  Gets the data date.

- **Returns** – the end date

- **getDurationFC**

  `public double getDurationFC()`

  - **Description**

    Gets the duration fc.

  - **Returns** – the duration fc

- **getEndDate**

  `public java.sql.Date getEndDate()`

  - **Description**

    Gets the end date.

  - **Returns** – the end date

- **getFcRowsDismissed**

  `public int getFcRowsDismissed()`

  - **Description**

    Gets the fc rows dismissed.

  - **Returns** – the fc rows dismissed

- **getFcRowsLoaded**

`public int getFcRowsLoaded()`

- **Description**

  Gets the fc rows loaded.

- **Returns** – the fc rows loaded

- **getFcRowsRead**

  `public int getFcRowsRead()`

  - **Description**

    Gets the fc rows read.

  - **Returns** – the fc rows read

- **getFcRowsRejected**

  `public int getFcRowsRejected()`

  - **Description**

    Gets the fc rows rejected.

  - **Returns** – the fc rows rejected

- **getFcRowsUpdated**

  `public int getFcRowsUpdated()`

  - **Description**

    Gets the fc rows updated.

  - **Returns** – the fc rows updated

- **getFinalRowID**

`public java.lang.String getFinalRowID()`

- **Description**

  Gets the final row id.

- **Returns** – the final row id

- **getOperationID**

  `public int getOperationID()`

  - **Description**

    Gets the operation id.

  - **Returns** – the operation id

- **getOperationRowID**

  `public java.lang.String getOperationRowID()`

  - **Description**

    Gets the operation row id.

  - **Returns** – the operation row id

- **getOperationType**

  `public java.lang.String getOperationType()`

  - **Description**

    Gets the operation type.

  - **Returns** – the operation type

- **getStartDate**

```
public java.sql.Date getStartDate()
```

- **Description**

  Gets the start date.

- **Returns** – the start date

- **getStatus**

```
public java.lang.String getStatus()
```

- **Description**

  Gets the status.

- **Returns** – the status

- **setDataDate**

```
public void setDataDate(java.sql.Date dataDate)
```

- **Description**

  Sets the data date.

- **Parameters**

  * `dataDate` – the new data date

- **setDurationFC**

```
public void setDurationFC(double durationFC)
```

- **Description**

  Sets the duration fc.

- **Parameters**

  * `durationFC` – the new duration fc

- **setEndDate**

  ```
  public void setEndDate(java.sql.Date endDate)
  ```

  - **Description**

    Sets the end date.

  - **Parameters**

    * `endDate` – the new end date

- **setFcRowsDismissed**

  ```
  public void setFcRowsDismissed(int fcRowsDismissed)
  ```

  - **Description**

    Sets the fc rows dismissed.

  - **Parameters**

    * `fcRowsDismissed` – the new fc rows dismissed

- **setFcRowsLoaded**

  ```
  public void setFcRowsLoaded(int fcRowsLoaded)
  ```

  - **Description**

    Sets the fc rows loaded.

  - **Parameters**

    * `fcRowsLoaded` – the new fc rows loaded

- **setFcRowsRead**

  ```
  public void setFcRowsRead(int fcRowsRead)
  ```

- **Description**

  Sets the fc rows read.

- **Parameters**

  * `fcRowsRead` – the new fc rows read

- **setFcRowsRejected**

```
public void setFcRowsRejected(int fcRowsRejected)
```

  - **Description**

    Sets the fc rows rejected.

  - **Parameters**

    * `fcRowsRejected` – the new fc rows rejected

- **setFcRowsUpdated**

```
public void setFcRowsUpdated(int fcRowsUpdated)
```

  - **Description**

    Sets the fc rows updated.

  - **Parameters**

    * `fcRowsUpdated` – the new fc rows updated

- **setFinalRowID**

```
public void setFinalRowID(java.lang.String finalRowID)
```

  - **Description**

    Sets the final row id.

  - **Parameters**

* `finalRowID` – the new final row id

- **setOperationID**

  ```
  public void setOperationID(int operationID)
  ```

  – **Description**

  Sets the operation id.

  – **Parameters**

  * `operationID` – the new operation id

- **setOperationRowID**

  ```
  public void setOperationRowID(java.lang.String operationRowID)
  ```

  – **Description**

  Sets the operation row id.

  – **Parameters**

  * `operationRowID` – the new operation row id

- **setOperationType**

  ```
  public void setOperationType(java.lang.String operationType)
  ```

  – **Description**

  Sets the operation type.

  – **Parameters**

  * `operationType` – the new operation type

- **setStartDate**

  ```
  public void setStartDate(java.sql.Date startDate)
  ```

- **Description**

  Sets the start date.

- **Parameters**

  * `startDate` – the new start date

- **setStatus**

```
public void setStatus(java.lang.String status)
```

  - **Description**

    Sets the status.

  - **Parameters**

    * `status` – the new status

- **toString**

```
public java.lang.String toString()
```

1.17 Class StatusEnum

The Enum StatusEnum. Holds all the possible values for the status of an execution or job.

Declaration

```
public final class StatusEnum
 extends java.lang.Enum
```

Field summary

CANCEL If the execution was canceled.
CHECK If the execution needs to be checked.
KO If the execution failed.
OK If the execution passed.
PDTE If the execution is waiting to be run.

Constructor summary

StatusEnum()

Method summary

valueOf(String)
values()

Fields

- `public static final StatusEnum` **OK**
  - If the execution passed.
- `public static final StatusEnum` **KO**
  - If the execution failed.
- `public static final StatusEnum` **CHECK**
  - If the execution needs to be checked.
- `public static final StatusEnum` **PDTE**
  - If the execution is waiting to be run.
- `public static final StatusEnum` **CANCEL**
  - If the execution was canceled.

Constructors

- **StatusEnum**

  private StatusEnum()

Methods

- **valueOf**

```
public static StatusEnum valueOf(java.lang.String name)
```

- **values**

```
public static StatusEnum[] values()
```

Members inherited from class Enum

`java.lang.Enum`

- `protected final Object` **clone**`() throws CloneNotSupportedException`
- `public final int` **compareTo**`(Enum` **arg0**`)`
- `public final boolean` **equals**`(Object` **arg0**`)`
- `protected final void` **finalize**`()`
- `public final Class` **getDeclaringClass**`()`
- `public final int` **hashCode**`()`
- `private final` **name**
- `public final String` **name**`()`
- `private final` **ordinal**
- `public final int` **ordinal**`()`
- `private void` **readObject**`(java.io.ObjectInputStream` **arg0**`) throws java.io.IOException, java.lang.ClassNotFoundException`
- `private void` **readObjectNoData**`() throws java.io.ObjectStreamException`
- `public String` **toString**`()`
- `public static Enum` **valueOf**`(Class` **arg0**`, String` **arg1**`)`

1.18 Class TechnicalCertificate

The Class TechnicalCertificate. Represents a technical certificate.

Declaration

```
public class TechnicalCertificate
 extends com.indra.iquality.model.Certificate
```

Field summary

details The details of the certificate.
numberOfRegisters The number of registers.

Constructor summary

TechnicalCertificate()

Method summary

equals(Object)
getDetails() Gets the details of the certificate.
getNumberOfRegisters() Gets the number of registers.
hashCode()
setDetails(TechnicalCertificateDetail) Sets the details of the certificate.
setNumberOfRegisters(int) Sets the number of registers.
toString()

Fields

- `private int` **numberOfRegisters**
  - The number of registers.
- `private TechnicalCertificateDetail` **details**
  - The details of the certificate.

Constructors

- **TechnicalCertificate**

```
public TechnicalCertificate()
```

Methods

- **equals**

  ```
  public boolean equals(java.lang.Object arg0)
  ```

- **getDetails**

  ```
  public TechnicalCertificateDetail getDetails()
  ```

  - **Description**

    Gets the details of the certificate.

  - **Returns** – the details

- **getNumberOfRegisters**

  ```
  public int getNumberOfRegisters()
  ```

  - **Description**

    Gets the number of registers.

  - **Returns** – the number of registers

- **hashCode**

  ```
  public native int hashCode()
  ```

- **setDetails**

  ```
  public void setDetails(TechnicalCertificateDetail details)
  ```

  - **Description**

    Sets the details of the certificate.

- **Parameters**
  - `details` – the new details

- **setNumberOfRegisters**

```
public void setNumberOfRegisters(int numRegistros)
```

  - **Description**

    Sets the number of registers.

  - **Parameters**
    - `numRegistros` – the new number of registers

- **toString**

```
public java.lang.String toString()
```

Members inherited from class Certificate

`com.indra.iquality.model.Certificate` (in 1.4, page 122)

- `protected` **certificate**
- `protected` **certificateDescription**
- `protected` **date**
- `protected` **entity**
- `public String` **getCertificate()**
- `public String` **getCertificateDescription()**
- `public String` **getDate()**
- `public String` **getEntity()**
- `public String` **getMetric()**
- `public String` **getMonth()**
- `public String` **getSection()**
- `public String` **getStatus()**
- `public String` **getSubsection()**

- `protected` **metric**
- `protected` **month**
- `protected` **section**
- `public void` **setCertificate**(`java.lang.String` **certificate**)
- `public void` **setCertificateDescription**(`java.lang.String` **certificateDescription**)
- `public void` **setDate**(`java.lang.String` **date**)
- `public void` **setEntity**(`java.lang.String` **entity**)
- `public void` **setIdMetrica**(`java.lang.String` **metric**)
- `public void` **setMonth**(`java.lang.String` **month**)
- `public void` **setSection**(`java.lang.String` **section**)
- `public void` **setStatus**(`java.lang.String` **status**)
- `public void` **setSubsection**(`java.lang.String` **subsection**)
- `protected` **status**
- `protected` **subsection**

1.19 Class TechnicalCertificateDetail

The Class DetailOfTechnicalCertificate. Represents the details of a technical certificate.

Declaration

```
public class TechnicalCertificateDetail
 extends java.lang.Object
```

Field summary

MAX_DIMENSIONES The maximum number of dimensions that a detail view will have.
valDimension1 The value of the dimension 1.
valDimension10 The value of the dimension 10.
valDimension11 The value of the dimension 11.
valDimension12 The value of the dimension 12.
valDimension13 The value of the dimension 13.
valDimension14 The value of the dimension 14.
valDimension15 The value of the dimension 15.
valDimension16 The value of the dimension 16.

valDimension17 The value of the dimension 17.
valDimension18 The value of the dimension 18.
valDimension19 The value of the dimension 19.
valDimension2 The value of the dimension 2.
valDimension20 The value of the dimension 20.
valDimension21 The value of the dimension 21.
valDimension22 The value of the dimension 22.
valDimension23 The value of the dimension 23.
valDimension24 The value of the dimension 24.
valDimension25 The value of the dimension 25.
valDimension3 The value of the dimension 3.
valDimension4 The value of the dimension 4.
valDimension5 The value of the dimension 5.
valDimension6 The value of the dimension 6.
valDimension7 The value of the dimension 7.
valDimension8 The value of the dimension 8.
valDimension9 The value of the dimension 9.

Constructor summary

TechnicalCertificateDetail(List) Instantiates a new detail of technical certificate.

Method summary

getValDimension1() Gets the val dimension1.
getValDimension10() Gets the val dimension10.
getValDimension11() Gets the val dimension11.
getValDimension12() Gets the val dimension12.
getValDimension13() Gets the val dimension13.
getValDimension14() Gets the val dimension14.
getValDimension15() Gets the val dimension15.
getValDimension16() Gets the val dimension16.
getValDimension17() Gets the val dimension17.
getValDimension18() Gets the val dimension18.
getValDimension19() Gets the val dimension19.
getValDimension2() Gets the val dimension2.
getValDimension20() Gets the val dimension20.
getValDimension21() Gets the val dimension21.
getValDimension22() Gets the val dimension22.
getValDimension23() Gets the val dimension23.
getValDimension24() Gets the val dimension24.

getValDimension25() Gets the val dimension25.
getValDimension3() Gets the val dimension3.
getValDimension4() Gets the val dimension4.
getValDimension5() Gets the val dimension5.
getValDimension6() Gets the val dimension6.
getValDimension7() Gets the val dimension7.
getValDimension8() Gets the val dimension8.
getValDimension9() Gets the val dimension9.
setValDimension1(String) Sets the val dimension1.
setValDimension10(String) Sets the val dimension10.
setValDimension11(String) Sets the val dimension11.
setValDimension12(String) Sets the val dimension12.
setValDimension13(String) Sets the val dimension13.
setValDimension14(String) Sets the val dimension14.
setValDimension15(String) Sets the val dimension15.
setValDimension16(String) Sets the val dimension16.
setValDimension17(String) Sets the val dimension17.
setValDimension18(String) Sets the val dimension18.
setValDimension19(String) Sets the val dimension19.
setValDimension2(String) Sets the val dimension2.
setValDimension20(String) Sets the val dimension20.
setValDimension21(String) Sets the val dimension21.
setValDimension22(String) Sets the val dimension22.
setValDimension23(String) Sets the val dimension23.
setValDimension24(String) Sets the val dimension24.
setValDimension25(String) Sets the val dimension25.
setValDimension3(String) Sets the val dimension3.
setValDimension4(String) Sets the val dimension4.
setValDimension5(String) Sets the val dimension5.
setValDimension6(String) Sets the val dimension6.
setValDimension7(String) Sets the val dimension7.
setValDimension8(String) Sets the val dimension8.
setValDimension9(String) Sets the val dimension9.

Fields

- `public static final int` **MAX_DIMENSIONES**
  - The maximum number of dimensions that a detail view will have.
- `private java.lang.String` **valDimension1**
  - The value of the dimension 1.

- `private java.lang.String` **valDimension2**
  - The value of the dimension 2.
- `private java.lang.String` **valDimension3**
  - The value of the dimension 3.
- `private java.lang.String` **valDimension4**
  - The value of the dimension 4.
- `private java.lang.String` **valDimension5**
  - The value of the dimension 5.
- `private java.lang.String` **valDimension6**
  - The value of the dimension 6.
- `private java.lang.String` **valDimension7**
  - The value of the dimension 7.
- `private java.lang.String` **valDimension8**
  - The value of the dimension 8.
- `private java.lang.String` **valDimension9**
  - The value of the dimension 9.
- `private java.lang.String` **valDimension10**
  - The value of the dimension 10.
- `private java.lang.String` **valDimension11**
  - The value of the dimension 11.
- `private java.lang.String` **valDimension12**
  - The value of the dimension 12.
- `private java.lang.String` **valDimension13**
  - The value of the dimension 13.
- `private java.lang.String` **valDimension14**

  - The value of the dimension 14.

- `private java.lang.String` **valDimension15**
  - The value of the dimension 15.
- `private java.lang.String` **valDimension16**
  - The value of the dimension 16.
- `private java.lang.String` **valDimension17**
  - The value of the dimension 17.
- `private java.lang.String` **valDimension18**
  - The value of the dimension 18.
- `private java.lang.String` **valDimension19**
  - The value of the dimension 19.
- `private java.lang.String` **valDimension20**
  - The value of the dimension 20.
- `private java.lang.String` **valDimension21**
  - The value of the dimension 21.
- `private java.lang.String` **valDimension22**
  - The value of the dimension 22.
- `private java.lang.String` **valDimension23**
  - The value of the dimension 23.
- `private java.lang.String` **valDimension24**
  - The value of the dimension 24.
- `private java.lang.String` **valDimension25**
  - The value of the dimension 25.

Constructors

- **TechnicalCertificateDetail**

  ```
  public TechnicalCertificateDetail(java.util.List values)
  ```

  - **Description**

    Instantiates a new detail of technical certificate.

  - **Parameters**

    * `values` – the values

Methods

- **getValDimension1**

  ```
  public java.lang.String getValDimension1()
  ```

  - **Description**

    Gets the val dimension1.

  - **Returns** – the val dimension1

- **getValDimension10**

  ```
  public java.lang.String getValDimension10()
  ```

  - **Description**

    Gets the val dimension10.

  - **Returns** – the val dimension10

- **getValDimension11**

  ```
  public java.lang.String getValDimension11()
  ```

- **Description**

  Gets the val dimension11.

- **Returns** – the val dimension11

- **getValDimension12**

  ```
  public java.lang.String getValDimension12()
  ```

  - **Description**

    Gets the val dimension12.

  - **Returns** – the val dimension12

- **getValDimension13**

  ```
  public java.lang.String getValDimension13()
  ```

  - **Description**

    Gets the val dimension13.

  - **Returns** – the val dimension13

- **getValDimension14**

  ```
  public java.lang.String getValDimension14()
  ```

  - **Description**

    Gets the val dimension14.

  - **Returns** – the val dimension14

- **getValDimension15**

  ```
  public java.lang.String getValDimension15()
  ```

- **Description**

  Gets the val dimension15.

- **Returns** – the val dimension15

- **getValDimension16**

  ```
  public java.lang.String getValDimension16()
  ```

  - **Description**

    Gets the val dimension16.

  - **Returns** – the val dimension16

- **getValDimension17**

  ```
  public java.lang.String getValDimension17()
  ```

  - **Description**

    Gets the val dimension17.

  - **Returns** – the val dimension17

- **getValDimension18**

  ```
  public java.lang.String getValDimension18()
  ```

  - **Description**

    Gets the val dimension18.

  - **Returns** – the val dimension18

- **getValDimension19**

  ```
  public java.lang.String getValDimension19()
  ```

- **Description**

  Gets the val dimension19.

- **Returns** – the val dimension19

- **getValDimension2**

  ```
  public java.lang.String getValDimension2()
  ```

  - **Description**

    Gets the val dimension2.

  - **Returns** – the val dimension2

- **getValDimension20**

  ```
  public java.lang.String getValDimension20()
  ```

  - **Description**

    Gets the val dimension20.

  - **Returns** – the val dimension20

- **getValDimension21**

  ```
  public java.lang.String getValDimension21()
  ```

  - **Description**

    Gets the val dimension21.

  - **Returns** – the val dimension21

- **getValDimension22**

  ```
  public java.lang.String getValDimension22()
  ```

- **Description**

  Gets the val dimension22.

- **Returns** – the val dimension22

- **getValDimension23**

  ```
  public java.lang.String getValDimension23()
  ```

  - **Description**

    Gets the val dimension23.

  - **Returns** – the val dimension23

- **getValDimension24**

  ```
  public java.lang.String getValDimension24()
  ```

  - **Description**

    Gets the val dimension24.

  - **Returns** – the val dimension24

- **getValDimension25**

  ```
  public java.lang.String getValDimension25()
  ```

  - **Description**

    Gets the val dimension25.

  - **Returns** – the val dimension25

- **getValDimension3**

  ```
  public java.lang.String getValDimension3()
  ```

- **Description**

  Gets the val dimension3.

- **Returns** – the val dimension3

- **getValDimension4**

  ```
  public java.lang.String getValDimension4()
  ```

  - **Description**

    Gets the val dimension4.

  - **Returns** – the val dimension4

- **getValDimension5**

  ```
  public java.lang.String getValDimension5()
  ```

  - **Description**

    Gets the val dimension5.

  - **Returns** – the val dimension5

- **getValDimension6**

  ```
  public java.lang.String getValDimension6()
  ```

  - **Description**

    Gets the val dimension6.

  - **Returns** – the val dimension6

- **getValDimension7**

  ```
  public java.lang.String getValDimension7()
  ```

- **Description**

  Gets the val dimension7.

- **Returns** – the val dimension7

- **getValDimension8**

  ```
  public java.lang.String getValDimension8()
  ```

  - **Description**

    Gets the val dimension8.

  - **Returns** – the val dimension8

- **getValDimension9**

  ```
  public java.lang.String getValDimension9()
  ```

  - **Description**

    Gets the val dimension9.

  - **Returns** – the val dimension9

- **setValDimension1**

  ```
  public void setValDimension1(java.lang.String valDimension1)
  ```

  - **Description**

    Sets the val dimension1.

  - **Parameters**

    * `valDimension1` – the new val dimension1

- **setValDimension10**

  ```
  public void setValDimension10(java.lang.String valDimension10)
  ```

- **Description**

  Sets the val dimension10.

- **Parameters**

  * `valDimension10` – the new val dimension10

- **setValDimension11**

```
public void setValDimension11(java.lang.String valDimension11)
```

  - **Description**

    Sets the val dimension11.

  - **Parameters**

    * `valDimension11` – the new val dimension11

- **setValDimension12**

```
public void setValDimension12(java.lang.String valDimension12)
```

  - **Description**

    Sets the val dimension12.

  - **Parameters**

    * `valDimension12` – the new val dimension12

- **setValDimension13**

```
public void setValDimension13(java.lang.String valDimension13)
```

  - **Description**

    Sets the val dimension13.

  - **Parameters**

* `valDimension13` – the new val dimension13

- **setValDimension14**

  ```
  public void setValDimension14(java.lang.String valDimension14)
  ```

  - **Description**

    Sets the val dimension14.

  - **Parameters**

    * `valDimension14` – the new val dimension14

- **setValDimension15**

  ```
  public void setValDimension15(java.lang.String valDimension15)
  ```

  - **Description**

    Sets the val dimension15.

  - **Parameters**

    * `valDimension15` – the new val dimension15

- **setValDimension16**

  ```
  public void setValDimension16(java.lang.String valDimension16)
  ```

  - **Description**

    Sets the val dimension16.

  - **Parameters**

    * `valDimension16` – the new val dimension16

- **setValDimension17**

  ```
  public void setValDimension17(java.lang.String valDimension17)
  ```

- **Description**

  Sets the val dimension17.

- **Parameters**

  * `valDimension17` – the new val dimension17

- **setValDimension18**

```
public void setValDimension18(java.lang.String valDimension18)
```

  - **Description**

    Sets the val dimension18.

  - **Parameters**

    * `valDimension18` – the new val dimension18

- **setValDimension19**

```
public void setValDimension19(java.lang.String valDimension19)
```

  - **Description**

    Sets the val dimension19.

  - **Parameters**

    * `valDimension19` – the new val dimension19

- **setValDimension2**

```
public void setValDimension2(java.lang.String valDimension2)
```

  - **Description**

    Sets the val dimension2.

  - **Parameters**

* `valDimension2` – the new val dimension2

- **setValDimension20**

  ```
  public void setValDimension20(java.lang.String valDimension20)
  ```

  - **Description**

    Sets the val dimension20.

  - **Parameters**

    * `valDimension20` – the new val dimension20

- **setValDimension21**

  ```
  public void setValDimension21(java.lang.String valDimension21)
  ```

  - **Description**

    Sets the val dimension21.

  - **Parameters**

    * `valDimension21` – the new val dimension21

- **setValDimension22**

  ```
  public void setValDimension22(java.lang.String valDimension22)
  ```

  - **Description**

    Sets the val dimension22.

  - **Parameters**

    * `valDimension22` – the new val dimension22

- **setValDimension23**

  ```
  public void setValDimension23(java.lang.String valDimension23)
  ```

- **Description**

  Sets the val dimension23.

- **Parameters**

  * `valDimension23` – the new val dimension23

- **setValDimension24**

```
public void setValDimension24(java.lang.String valDimension24)
```

  - **Description**

    Sets the val dimension24.

  - **Parameters**

    * `valDimension24` – the new val dimension24

- **setValDimension25**

```
public void setValDimension25(java.lang.String valDimension25)
```

  - **Description**

    Sets the val dimension25.

  - **Parameters**

    * `valDimension25` – the new val dimension25

- **setValDimension3**

```
public void setValDimension3(java.lang.String valDimension3)
```

  - **Description**

    Sets the val dimension3.

  - **Parameters**

* `valDimension3` – the new val dimension3

- **setValDimension4**

  public void setValDimension4(java.lang.String valDimension4)

  - **Description**

    Sets the val dimension4.

  - **Parameters**

    * `valDimension4` – the new val dimension4

- **setValDimension5**

  public void setValDimension5(java.lang.String valDimension5)

  - **Description**

    Sets the val dimension5.

  - **Parameters**

    * `valDimension5` – the new val dimension5

- **setValDimension6**

  public void setValDimension6(java.lang.String valDimension6)

  - **Description**

    Sets the val dimension6.

  - **Parameters**

    * `valDimension6` – the new val dimension6

- **setValDimension7**

  public void setValDimension7(java.lang.String valDimension7)

- **Description**

  Sets the val dimension7.

- **Parameters**

  * `valDimension7` – the new val dimension7

- **setValDimension8**

```
public void setValDimension8(java.lang.String valDimension8)
```

  - **Description**

    Sets the val dimension8.

  - **Parameters**

    * `valDimension8` – the new val dimension8

- **setValDimension9**

```
public void setValDimension9(java.lang.String valDimension9)
```

  - **Description**

    Sets the val dimension9.

  - **Parameters**

    * `valDimension9` – the new val dimension9

2 Package com.indra.iquality.controller

Package Contents *Page*

Classes

Provides the controllers that will handle the HTTP requests to the web application.

2.1 Class APIController

The Class APIController.

Declaration

```
public class APIController
 extends java.lang.Object
```

Field summary

DICTIONARY_CACHE_FILE The Constant representing the path where to save the verbose (extra) cache file.
DICTIONARY_JSON_CACHE_FILE The Constant representing the path where to save the necessary cache file.
environment The Constant reference to the environment.
logger The Constant logger.
VERBOSE_CACHE Sets the level of the cache to verbose or not.

Constructor summary

APIController()

Method summary

auxiliaryUpdateDictionaryCache() Auxiliary method to handle the actual update dictionary cache.

auxiliaryWriteQueryResultToTextFile(List, String) Updates the verbose cache, where each line of the file represented by sourcePath contains a record of the query.

getComponentDescription(String, String) Gets the description of a component of the dictionary.

getDictionaryAsJSONTree() Gets the JSON representation of the tree corresponding to the dictionary of concepts.

getJobDependencies(int, String) Gets the dependencies of a given job represented as a JSONArray.

getOperationTrace(int) Gets the JSONArray representation of the trace of an operation.

updateDictionaryCache() Updates the cache representation of the dictionary.

Fields

- `private static final org.slf4j.Logger` **logger**
  - The Constant logger.
- `private static final com.indra.iquality.singleton.Environment` **environment**
  - The Constant reference to the environment.
- `private static final java.lang.String` **DICTIONARY_CACHE_FILE**
  - The Constant representing the path where to save the verbose (extra) cache file.
- `private static final java.lang.String` **DICTIONARY_JSON_CACHE_FILE**
  - The Constant representing the path where to save the necessary cache file.
- `private static boolean` **VERBOSE_CACHE**
  - Sets the level of the cache to verbose or not. In verbose mode an extra cache file is saved, with the result of the query where the rows are saved as lines in the file.

Constructors

- **APIController**

```
public APIController()
```

Methods

- **auxiliaryUpdateDictionaryCache**

  ```
  private boolean auxiliaryUpdateDictionaryCache()
  ```

  - **Description**

    Auxiliary method to handle the actual update dictionary cache. Used by `updateDictionaryCache()`

  - **Returns** – true, if successful

- **auxiliaryWriteQueryResultToTextFile**

  ```
  private boolean auxiliaryWriteQueryResultToTextFile(java.util.
      List conceptNodes,java.lang.String sourcePath)
  ```

  - **Description**

    Updates the verbose cache, where each line of the file represented by sourcePath contains a record of the query.

  - **Parameters**

    * `conceptNodes` – the node representation of the nodes to store

    * `sourcePath` – the path where to store the verbose cache

  - **Returns** – true, if successful

- **getComponentDescription**

  ```
  private org.json.simple.JSONObject getComponentDescription(java.
      lang.String type,java.lang.String idComponente)
  ```

- **Description**

  Gets the description of a component of the dictionary.

- **Parameters**

  * `type` – the type of the component

  * `idComponente` – the identifier of the component. The following format is expected: **compRowID:<the row id of the component>&ctRowID:<the row id of the ct>**

- **Returns** – the JSON representation of the description of the component

- **getDictionaryAsJSONTree**

```
private org.json.simple.JSONObject getDictionaryAsJSONTree()
```

  - **Description**

    Gets the JSON representation of the tree corresponding to the dictionary of concepts. The JSON is read from a cache file located at `DICTIONARY_JSON_CACHE_FILE` (in 2.1, page 229). The format of the JSON is that expected by the **jsTree** plug-in for jQuery.

  - **Returns** – the representation of the dictionary as JSON

- **getJobDependencies**

```
private org.json.simple.JSONArray getJobDependencies(int
    idEjecucion,java.lang.String idJob)
```

  - **Description**

    Gets the dependencies of a given job represented as a JSONArray.

  - **Parameters**

    * `idEjecucion` – the identifier of the execution where the job takes part

    * `idJob` – the identifier of the job

  - **Returns** – the dependencies of the given job

- **getOperationTrace**

  ```
  private org.json.simple.JSONArray getOperationTrace(int
      idOperacion)
  ```

  - **Description**

    Gets the JSONArray representation of the trace of an operation.

  - **Parameters**

    * `idOperacion` – the identifier of the operation

  - **Returns** – the operation trace as a JSONArray

- **updateDictionaryCache**

  ```
  private java.lang.String updateDictionaryCache()
  ```

  - **Description**

    Updates the cache representation of the dictionary. Should be called each time a change is done in the representation of the dictionary. Relies heavily on `auxiliaryUpdateDictionaryCache(`
    . When the verbose level is set and the cache gets updated, also updates the verbose cache by calling `auxiliaryWriteQueryResultToTextFile(List, String)`

  - **Returns** – a redirection to the main page

2.2 Class BaseController

The Class BaseController.

Declaration

```
public class BaseController
 extends java.lang.Object
```

Field summary

logger The Constant logger.

VIEW_ERROR The Constant pointing to the default error page.
VIEW_INDEX The Constant pointing to the main page of the application.
VIEW_LOGIN The Constant pointing to the login view.
VIEW_NOT_FOUND The Constant pointing to the default view when a page was not found.
VIEW_REGISTRATION The Constant pointing to the registration view.

Constructor summary

BaseController()

Method summary

notFound(ModelMap) Handles a GET request to display the default view when a page was not found.
serverError(ModelMap) Handles a GET request to display the default error page.
showLogin(ModelMap) Handles a GET request to display the login view.
showMain(ModelMap) Handles a GET request to display the main page of the application.
showRegistration(ModelMap) Handles a GET request to display the registration view.

Fields

- `private static final java.lang.String` **VIEW_INDEX**
  - The Constant pointing to the main page of the application.
- `private static final java.lang.String` **VIEW_LOGIN**
  - The Constant pointing to the login view.
- `private static final java.lang.String` **VIEW_REGISTRATION**
  - The Constant pointing to the registration view.
- `private static final java.lang.String` **VIEW_NOT_FOUND**
  - The Constant pointing to the default view when a page was not found.

- `private static final java.lang.String` **VIEW_ERROR**
  - The Constant pointing to the default error page.
- `private static final org.slf4j.Logger` **logger**
  - The Constant logger.

Constructors

- **BaseController**

```
public BaseController()
```

Methods

- **notFound**

```
private java.lang.String notFound(org.springframework.ui.
    ModelMap model)
```

  - **Description**

    Handles a GET request to display the default view when a page was not found.

  - **Parameters**
    - `model` – the model to pass data to the view
  - **Returns** – the view to display

- **serverError**

```
private java.lang.String serverError(org.springframework.ui.
    ModelMap model)
```

  - **Description**

    Handles a GET request to display the default error page.

- **Parameters**
  - `model` – the model to pass data to the view
- **Returns** – the view to display

- **showLogin**

```
private java.lang.String showLogin(org.springframework.ui.
    ModelMap model)
```

  - **Description**

    Handles a GET request to display the login view.
  - **Parameters**
    - `model` – the model to pass data to the view
  - **Returns** – the view to display

- **showMain**

```
private java.lang.String showMain(org.springframework.ui.
    ModelMap model)
```

  - **Description**

    Handles a GET request to display the main page of the application.
  - **Parameters**
    - `model` – the model to pass data to the view
  - **Returns** – the view to display

- **showRegistration**

```
private java.lang.String showRegistration(org.springframework.ui
    .ModelMap model)
```

– **Description**

Handles a GET request to display the registration view.

– **Parameters**

* `model` – the model to pass data to the view

– **Returns** – the view to display

2.3 Class CertificatesResultController

The Class CertificatesResultController. Handles all the requests related to the management (mainly reads) related with certificates.

Declaration

```
public class CertificatesResultController
 extends java.lang.Object
```

Field summary

environment The Constant reference to the environment.
logger The Constant logger.
TAB_VAL_BUSSINESS_CERTIFICATE The Constant to represent the first tab.
TAB_VAL_TECHNICAL_CERTIFICATE The Constant to represent the second tab.
VIEW_BUSINESS_CERTIFICATES The Constant pointing to the view of all the business certificates.
VIEW_BUSINESS_CERTIFICATES_DETAIL The Constant pointing to a detailed view of a business certificates.
VIEW_TECHNICAL_CERTIFICATES The Constant pointing to the view of all the technical certificates.
VIEW_TECHNICAL_CERTIFICATES_DETAIL The Constant pointing to a detailed view of a technical certificates.

Constructor summary

CertificatesResultController()

Method summary

getCertificates(int, ModelMap) Handles a GET request to display all the certificates of a given type for the current system and software version.

getDetailsOfCertificate(int, String, String, ModelMap, HttpServletResponse) Handles a GET request to display the detailed view of a certificate of a given type for the current system and software version.

Fields

- `private static final org.slf4j.Logger` **logger**
  - The Constant logger.
- `private static final com.indra.iquality.singleton.Environment` **environment**
  - The Constant reference to the environment.
- `private static final java.lang.String` **VIEW_BUSINESS_CERTIFICATES**
  - The Constant pointing to the view of all the business certificates.
- `private static final java.lang.String` **VIEW_BUSINESS_CERTIFICATES_DETAIL**
  - The Constant pointing to a detailed view of a business certificates.
- `private static final java.lang.String` **VIEW_TECHNICAL_CERTIFICATES**
  - The Constant pointing to the view of all the technical certificates.
- `private static final java.lang.String` **VIEW_TECHNICAL_CERTIFICATES_DETAIL**
  - The Constant pointing to a detailed view of a technical certificates.
- `private static final int` **TAB_VAL_BUSSINESS_CERTIFICATE**
  - The Constant to represent the first tab.
- `private static final int` **TAB_VAL_TECHNICAL_CERTIFICATE**
  - The Constant to represent the second tab.

Constructors

- **CertificatesResultController**

```
public CertificatesResultController()
```

Methods

- **getCertificates**

```
private java.lang.String getCertificates(int tab,org.
    springframework.ui.ModelMap model)
```

  - **Description**

    Handles a GET request to display all the certificates of a given type for the current system and software version.

  - **Parameters**

    * `tab` – the type of certificates to view; currently supports `TAB_VAL_BUSSINESS_CERTIFICATE` (in 2.3, page 237) and `TAB_VAL_TECHNICAL_CERTIFICATE` (in 2.3, page 237).

    * `model` – the model to pass data to the view

  - **Returns** – the view to display

- **getDetailsOfCertificate**

```
private java.lang.String getDetailsOfCertificate(int tab,java.
    lang.String idMetrica,java.lang.String idMes,org.
    springframework.ui.ModelMap model,javax.servlet.http.
    HttpServletResponse response)
```

  - **Description**

    Handles a GET request to display the detailed view of a certificate of a given type for the current system and software version. As the detailed tables have a variable

number of columns, this method leaves a cookie in the browser so that the frontend knows how many columns to display.

– **Parameters**

* `tab` – the type of certificates to view; currently supports `TAB_VAL_BUSSINESS_CERTIFICATE` (in 2.3, page 237) and `TAB_VAL_TECHNICAL_CERTIFICATE` (in 2.3, page 237).

* `idMetrica` – the metrics for which to check the certificate; submit as part of the query string

* `idMes` – the month for which to check the certificate; submit as part of the query string

* `model` – the model to pass data to the view

* `response` – needed to add cookies

– **Returns** – the view to display

2.4 Class DictionaryController

The Class DictionaryController. Handles the requests to display the dictionary of concepts. See `APIController` (in 2.1, page 228) if you don't see some methods related to the dictionary of concepts that you might have expected to find here.

Declaration

```
public class DictionaryController
 extends java.lang.Object
```

Field summary

logger The Constant logger.
VIEW_DICTIONARY The Constant pointing to the dictionary of concepts.

Constructor summary

DictionaryController()

Method summary

showDictionary(ModelMap) Handles a GET request to display the dictionary of concepts.

Fields

- `private static final org.slf4j.Logger` **logger**
  - The Constant logger.
- `private static final java.lang.String` **VIEW_DICTIONARY**
  - The Constant pointing to the dictionary of concepts.

Constructors

- **DictionaryController**

  ```
  public DictionaryController()
  ```

Methods

- **showDictionary**

  ```
  private java.lang.String showDictionary(org.springframework.ui.
      ModelMap model)
  ```

  - **Description**

    Handles a GET request to display the dictionary of concepts.

  - **Parameters**
    - `model` – the model to pass data to the view
  - **Returns** – the view to display

2.5 Class ExecutionManagementController

The Class ExecutionManagementController. Handles all the requests related to the management (mainly reads) related with executions of ETL flows, as well as its corresponding jobs and register of operations.

Declaration

```
public class ExecutionManagementController
 extends java.lang.Object
```

Field summary

environment The Constant reference to the environment.
logger The Constant logger.
VIEW_EXECUTIONS The Constant pointing to the view of all the executions.
VIEW_JOBS The Constant pointing to the view of all the jobs of an execution.
VIEW_REGOPS The Constant pointing to the view of the register of operations of a job.

Constructor summary

ExecutionManagementController()

Method summary

getJobsOfExecution(int, ModelMap) Handles a GET request to display all the jobs of a given execution for the current system and software version.
getRegopsOfJob(int, String, ModelMap) Handles a GET request to display the register of operations of a given job, which in turn is part of a given execution, for the current system and software version.
showAllExecutions(ModelMap) Handles a GET request to display all the executions for the current system and software version.
showExecution(int, ModelMap) Handles a GET request to display a single given execution for the current system and software version.

Fields

- `private static final org.slf4j.Logger` **logger**
  - The Constant logger.
- `private static final com.indra.iquality.singleton.Environment` **environment**
  - The Constant reference to the environment.
- `private static final java.lang.String` **VIEW_EXECUTIONS**
  - The Constant pointing to the view of all the executions.
- `private static final java.lang.String` **VIEW_JOBS**
  - The Constant pointing to the view of all the jobs of an execution.
- `private static final java.lang.String` **VIEW_REGOPS**
  - The Constant pointing to the view of the register of operations of a job.

Constructors

- **ExecutionManagementController**

```
public ExecutionManagementController()
```

Methods

- **getJobsOfExecution**

```
private java.lang.String getJobsOfExecution(int idEjecucion,org.
    springframework.ui.ModelMap model)
```

  - **Description**

    Handles a GET request to display all the jobs of a given execution for the current system and software version.

  - **Parameters**

    * `idEjecucion` – the unique identifier of the execution for which to display the jobs

    * `model` – the model to pass data to the view

  – **Returns** – the view to display

- **getRegopsOfJob**

```
private java.lang.String getRegopsOfJob(int idEjecucion,java.
    lang.String idJob,org.springframework.ui.ModelMap model)
```

  – **Description**

    Handles a GET request to display the register of operations of a given job, which in turn is part of a given execution, for the current system and software version.

  – **Parameters**

    * `idEjecucion` – the unique identifier of the execution to which the job belongs

    * `idJob` – the unique identifier of the for which to display the register of operations

    * `model` – the model to pass data to the view

  – **Returns** – the view to display

- **showAllExecutions**

```
private java.lang.String showAllExecutions(org.springframework.
    ui.ModelMap model)
```

  – **Description**

    Handles a GET request to display all the executions for the current system and software version.

  – **Parameters**

    * `model` – the model to pass data to the view

  – **Returns** – the view to display

- **showExecution**

```
private java.lang.String showExecution(int idEjecucion,org.
    springframework.ui.ModelMap model)
```

  - **Description**

    Handles a GET request to display a single given execution for the current system and software version.

  - **Parameters**

    * `idEjecucion` – the unique identifier of the execution to display

    * `model` – the model to pass data to the view

  - **Returns** – the view to display

2.6 Class FileLoadController

Declaration

```
public class FileLoadController
 extends java.lang.Object
```

Field summary

ACCEPTED_TYPE
environment The Constant reference to the environment.
logger The Constant logger.
VIEW_LOAD The Constant pointing to the view of all the executions.

Constructor summary

FileLoadController()

Method summary

handleFileUpload(MultipartHttpServletRequest, HttpServletResponse)
loadFiles(Model)

Fields

- `private static final org.slf4j.Logger` **logger**
  - The Constant logger.
- `private static final com.indra.iquality.singleton.Environment` **environment**
  - The Constant reference to the environment.
- `private static final java.lang.String` **ACCEPTED_TYPE**
- `private static final java.lang.String` **VIEW_LOAD**
  - The Constant pointing to the view of all the executions.

Constructors

- **FileLoadController**

```
public FileLoadController()
```

Methods

- **handleFileUpload**

```
public org.json.simple.JSONObject handleFileUpload(org.
    springframework.web.multipart.MultipartHttpServletRequest
    request,javax.servlet.http.HttpServletResponse response)
    throws java.io.IOException
```

- **loadFiles**

```
private java.lang.String loadFiles(org.springframework.ui.Model
    model)
```

2.7 Class FlowManagementController

The Class FlowManagmentController. Handles all the requests related to the management (CRUD operations) of ETL flows.

Declaration

```
public class FlowManagementController
 extends java.lang.Object
```

Field summary

environment The Constant reference to the environment.
logger The Constant logger.
VIEW_FLOWS The Constant pointing to the view of all the flows.
VIEW_WIZARD The Constant pointing to the view of the wizard for creating a new flow.

Constructor summary

FlowManagementController()

Method summary

handleNewFlowPost(String) Handle a POST request to create a new flow.
showAllFlows(Model) Handles a GET request to display all the flows for the current system and software version.
wizardNewFlow(Model) Handles a GET request to display a wizard for creating a new flow.

Fields

- `private static final org.slf4j.Logger` **logger**

  - The Constant logger.

- `private static final com.indra.iquality.singleton.Environment` **environment**
  - The Constant reference to the environment.
- `private static final java.lang.String` **VIEW_FLOWS**
  - The Constant pointing to the view of all the flows.
- `private static final java.lang.String` **VIEW_WIZARD**
  - The Constant pointing to the view of the wizard for creating a new flow.

Constructors

- **FlowManagementController**

```
public FlowManagementController()
```

Methods

- **handleNewFlowPost**

```
private org.json.simple.JSONObject handleNewFlowPost(java.lang.
    String jsonString)
```

  - **Description**

    Handle a POST request to create a new flow.

  - **Parameters**

    * `jsonString` – the json representation of the flow; contains a name *nombrePase*, a boolean string *esAtipico*, an array of job identifiers with at least one element *jobs*, and an object containing the dependencies *dependencias* as key-value pairs, where the values are arrays.

  - **Returns** – the representation of the response; contains at least the key *redirect*, which indicates the path to redirect the browser if the insert was successful. If the

handler catches an error will return a *error* key, with the path to redirect in case of failure.

- **showAllFlows**

  ```
  private java.lang.String showAllFlows(org.springframework.ui.
      Model model)
  ```

  - **Description**

    Handles a GET request to display all the flows for the current system and software version.

  - **Parameters**

    * `model` – the model to pass data to the view

  - **Returns** – the view to display

- **wizardNewFlow**

  ```
  private java.lang.String wizardNewFlow(org.springframework.ui.
      Model model)
  ```

  - **Description**

    Handles a GET request to display a wizard for creating a new flow.

  - **Parameters**

    * `model` – the model to pass data to the view

  - **Returns** – the view to display

2.8 Class ParametrizeCertificatesController

Declaration

```
public class ParametrizeCertificatesController
 extends java.lang.Object
```

Field summary

environment The Constant reference to the environment.
logger The Constant logger.
TAB_VAL_BUSSINESS_CERTIFICATE The Constant to represent the first tab.
TAB_VAL_TECHNICAL_CERTIFICATE The Constant to represent the second tab.
VIEW_PARAM_BUSINESS_CERTIFICATES The Constant pointing to the view to parametrize all the business certificates.
VIEW_PARAM_TECHNICAL_CERTIFICATES The Constant pointing to the view to parametrize the technical certificates.
VIEW_WIZARD The Constant pointing to the view to parametrize the technical certificates.

Constructor summary

ParametrizeCertificatesController()

Method summary

getParamCertificates(int, ModelMap)
handleNewCertificatePost(String)
wizardNewCertificate(Model)

Fields

- `private static final org.slf4j.Logger` **logger**
  - The Constant logger.
- `private static final com.indra.iquality.singleton.Environment` **environment**
  - The Constant reference to the environment.
- `private static final java.lang.String` **VIEW_PARAM_BUSINESS_CERTIFICATES**
  - The Constant pointing to the view to parametrize all the business certificates.

- `private static final java.lang.String` **VIEW_PARAM_TECHNICAL_CERTIFICATES**
  - The Constant pointing to the view to parametrize the technical certificates.
- `private static final java.lang.String` **VIEW_WIZARD**
  - The Constant pointing to the view to parametrize the technical certificates.
- `private static final int` **TAB_VAL_BUSSINESS_CERTIFICATE**
  - The Constant to represent the first tab.
- `private static final int` **TAB_VAL_TECHNICAL_CERTIFICATE**
  - The Constant to represent the second tab.

Constructors

- **ParametrizeCertificatesController**

```
public ParametrizeCertificatesController()
```

Methods

- **getParamCertificates**

```
private java.lang.String getParamCertificates(int tab,org.
    springframework.ui.ModelMap model)
```

- **handleNewCertificatePost**

```
private org.json.simple.JSONObject handleNewCertificatePost(java
    .lang.String jsonString)
```

- **wizardNewCertificate**

```
private java.lang.String wizardNewCertificate(org.
    springframework.ui.Model model)
```

3 Package com.indra.iquality.helper

Package Contents *Page*

Classes

Provides the helper modules needed all across the application.

3.1 Class DataHelper

The Class DataHelper. Provides a set of auxiliary methods related to data filtering and conversion needed for the DAOs.

Declaration

```
public class DataHelper
 extends java.lang.Object
```

Constructor summary

DataHelper()

Method summary

conceptTypeStringToEnum(String) Converts a String to its equivalent representation as one of the concept type defined in the system, if possible.

filterNullString(String) Filters a string in case it is null.

filterStringToDouble(String) Filters a string in case it is null an returns its representation as a double.

filterStringToInt(String) Filters a string in case it is null an returns its representation as an integer.

filterStringToSqlDate(String) Translates a String representation of a timestamp to an SQLDate.

Constructors

- **DataHelper**

```
public DataHelper()
```

Methods

- **conceptTypeStringToEnum**

```
public com.indra.iquality.model.ConceptTypeEnum
    conceptTypeStringToEnum(java.lang.String type)
```

  - **Description**

    Converts a String to its equivalent representation as one of the concept type defined in the system, if possible.

  - **Parameters**

    * `type` – the type of the concept

  - **Returns** – the concept translated to the system representation, or the default ünknown"value if no possible translation.

  - **See also**

    * `com.indra.iquality.model.ConceptTypeEnum` (in 1.7, page 138)

- **filterNullString**

```
public java.lang.String filterNullString(java.lang.String s)
```

  - **Description**

    Filters a string in case it is null.

- **Parameters**
  - `s` – the String
- **Returns** – if the String is null or holds the value "null", returns the default `Environment` (in 4.1, page 263) definition for a null String. Otherwise, returns the input parameter.

- **filterStringToDouble**

```
public double filterStringToDouble(java.lang.String s)
```

  - **Description**

    Filters a string in case it is null an returns its representation as a double.

  - **Parameters**
    - `s` – the String
  - **Returns** – if the String is null or holds the value "null", returns the default `Environment` (in 4.1, page 263) definition for a null double. Otherwise, returns the double representation of the input parameter.

- **filterStringToInt**

```
public int filterStringToInt(java.lang.String s)
```

  - **Description**

    Filters a string in case it is null an returns its representation as an integer.

  - **Parameters**
    - `s` – the String
  - **Returns** – if the String is null or holds the value "null", returns the default `Environment` (in 4.1, page 263) definition for a null integer. Otherwise, returns the integer representation of the input parameter.

- **filterStringToSqlDate**

```
public java.sql.Date filterStringToSqlDate(java.lang.String
    tmstmp) throws java.text.ParseException
```

- **Description**

  Translates a String representation of a timestamp to an SQLDate.

- **Parameters**

  * `tmstmp` – the timestamp

- **Returns** – the parsed date, or the default date defined in the `Environment` (in 4.1, page 263) for a null String

- **Throws**

  * `java.text.ParseException` – if the String was not formatted as a timestamp

3.2 Class GenericTreeNode

The Class GenericTreeNode. Represents a tree node of a generic type.

Declaration

```
public class GenericTreeNode
 extends java.lang.Object
```

Field summary

children The children.
data The data to store in the node.
parent The parent.

Constructor summary

GenericTreeNode() Instantiates a new generic tree node.
GenericTreeNode(T) Instantiates a new generic tree node with a given data.

Method summary

addChild(GenericTreeNode) Adds a child at the end of the current children.
addChildAt(int, GenericTreeNode) Adds a new child at a given position.
equals(Object)
getChildAt(int) Gets the child at a given index.
getChildren() Gets the children.
getData() Gets the data of the node.
getNumberOfChildren() Gets the number of children.
getParent() Gets the parent.
hasChildren() Checks if the node has children.
hashCode()
myPrintTree(int) Recursively prints the whole tree hanging from the node in a verbose manner: the data, the level and the children.
removeChildAt(int) Removes the child at the given index.
removeChildren() Removes all the children.
setChildren(List) Sets the children.
setData(T) Sets the data of the node.
toString()
toStringVerbose() Gets a verbose representation of the node, i. e., the data and all the children.

Fields

- `private java.lang.Object` **data**
  - The data to store in the node.
- `private java.util.List` **children**
  - The children.
- `private GenericTreeNode` **parent**
  - The parent.

Constructors

- **GenericTreeNode**

```
public GenericTreeNode()
```

- **Description**

  Instantiates a new generic tree node.

- **GenericTreeNode**

  ```
  public GenericTreeNode(java.lang.Object data)
  ```

  - **Description**

    Instantiates a new generic tree node with a given data.

  - **Parameters**

    * `data` – the data

Methods

- **addChild**

  ```
  public void addChild(GenericTreeNode child)
  ```

  - **Description**

    Adds a child at the end of the current children.

  - **Parameters**

    * `child` – the child

- **addChildAt**

  ```
  public void addChildAt(int index,GenericTreeNode child) throws
      java.lang.IndexOutOfBoundsException
  ```

  - **Description**

    Adds a new child at a given position.

  - **Parameters**

    * `index` – the index where to insert

    * `child` – the child

  - **Throws**

    * `java.lang.IndexOutOfBoundsException` – if the the insertion was meant outside the children

- **equals**

  ```
  public boolean equals(java.lang.Object arg0)
  ```

- **getChildAt**

  ```
  public GenericTreeNode getChildAt(int index) throws java.lang.
      IndexOutOfBoundsException
  ```

  - **Description**

    Gets the child at a given index.

  - **Parameters**

    * `index` – the index

  - **Returns** – the child at the given index

  - **Throws**

    * `java.lang.IndexOutOfBoundsException` – if the index is out of the bounds of chilrden

- **getChildren**

  ```
  public java.util.List getChildren()
  ```

  - **Description**

    Gets the children.

  - **Returns** – the children

- **getData**

  ```
  public java.lang.Object getData()
  ```

  - **Description**

    Gets the data of the node.

  - **Returns** – the data

- **getNumberOfChildren**

  ```
  public int getNumberOfChildren()
  ```

  - **Description**

    Gets the number of children.

  - **Returns** – the number of children

- **getParent**

  ```
  public GenericTreeNode getParent()
  ```

  - **Description**

    Gets the parent.

  - **Returns** – the parent

- **hasChildren**

  ```
  public boolean hasChildren()
  ```

  - **Description**

    Checks if the node has children.

  - **Returns** – true, if successful

- **hashCode**

  ```
  public native int hashCode()
  ```

- **myPrintTree**

  ```
  public void myPrintTree(int level) throws javax.activation.
      UnsupportedDataTypeException
  ```

  - **Description**

    Recursively prints the whole tree hanging from the node in a verbose manner: the data, the level and the children. It only works when the type T of the node is `DictionaryConcept` (in 1.9, page 147).

  - **Parameters**

    * `level` – the level of the node

  - **Throws**

    * `javax.activation.UnsupportedDataTypeException` – if the data type of the node is not `DictionaryConcept` (in 1.9, page 147)

- **removeChildAt**

  ```
  public void removeChildAt(int index) throws java.lang.
      IndexOutOfBoundsException
  ```

  - **Description**

    Removes the child at the given index.

  - **Parameters**

    * `index` – the index to remove

  - **Throws**

    * `java.lang.IndexOutOfBoundsException` – if the the removal was meant outside the children

- **removeChildren**

  ```
  public void removeChildren()
  ```

  - **Description**

    Removes all the children.

- **setChildren**

  ```
  public void setChildren(java.util.List children)
  ```

  - **Description**

    Sets the children.

  - **Parameters**

    * `children` – the new children

- **setData**

  ```
  public void setData(java.lang.Object data)
  ```

  - **Description**

    Sets the data of the node.

  - **Parameters**

    * `data` – the new data

- **toString**

  ```
  public java.lang.String toString()
  ```

- **toStringVerbose**

  ```
  public java.lang.String toStringVerbose()
  ```

– **Description**

Gets a verbose representation of the node, i. e., the data and all the children.

– **Returns** – the verbose string representation of the node

3.3 Class TreeTranslatorHelper

The Class TreeTranslatorHelper. Offers utilities to transform lists to trees, and trees to JSON.

Declaration

```
public class TreeTranslatorHelper
 extends java.lang.Object
```

Constructor summary

TreeTranslatorHelper()

Method summary

conceptListToTree(List) Translates a list of nodes with `DictionaryConcept` (in 1.9, page 147) as its data to a tree.
treeToJSONForjsTree(GenericTreeNode) Translates a tree of `DictionaryConcept` (in 1.9, page 147) to a JSONObject.
treeToPrettyJSONStringForjsTree(GenericTreeNode) Translates a tree of `DictionaryConcept` (in 1.9, page 147) to its pretty JSON String.

Constructors

- **TreeTranslatorHelper**

```
public TreeTranslatorHelper()
```

Methods

- **conceptListToTree**

  ```
  public GenericTreeNode conceptListToTree(java.util.List list)
  ```

  - **Description**

    Translates a list of nodes with `DictionaryConcept` (in 1.9, page 147) as its data to a tree.

  - **Parameters**

    * `list` – the list of nodes

  - **Returns** – the root node of the equivalent tree

- **treeToJSONForjsTree**

  ```
  public org.json.simple.JSONObject treeToJSONForjsTree(
      GenericTreeNode root)
  ```

  - **Description**

    Translates a tree of `DictionaryConcept` (in 1.9, page 147) to a JSONObject. The format of the JSON tree is the one expected by the **jsTree plug-in for jQuery**.

  - **Parameters**

    * `root` – the root node of the tree

  - **Returns** – the JSONObject representation of the tree

- **treeToPrettyJSONStringForjsTree**

  ```
  public java.lang.String treeToPrettyJSONStringForjsTree(
      GenericTreeNode root)
  ```

  - **Description**

Translates a tree of `DictionaryConcept` (in 1.9, page 147) to its pretty JSON String. That is, a friendly and natural formatted JSON. The format of the JSON tree is the one expected by the **jsTree plug-in for jQuery**.

– **Parameters**

* `root` – the root node of the tree

– **Returns** – the pretty JSON representation of the tree

4 Package com.indra.iquality.singleton

Package Contents *Page*

Classes

The Singleton Environment.

Provides the Singletons for the application.

4.1 Class Environment

The Singleton Environment. Defines the shared environment settings for all the modules of the application. Provides consistency across the application.

Declaration

```
public class Environment
 extends java.lang.Object
```

Field summary

currentSoftwareDescription A short description of the current software version.
currentSoftwareID The identifier of the current software version running in the system.
DEFAULT_NULL_DATE The default value to substitute when a null Date is found.

DEFAULT_NULL_DOUBLE The default value to substitute when a null double is found.
DEFAULT_NULL_INT The default value to substitute when a null integer is found.
DEFAULT_NULL_STRING The default value to substitute when a null String is found.
instance The unique instance of the environment.
softwareVersions Other software versions of the system.
systemDescription A short description of the underlying system.
systemID The identifier of the underlying system.

Constructor summary

Environment() Instantiates a new environment with the default values for the system and software.
Environment(String, String, int) Instantiates a new environment for a given system and first software version.

Method summary

getCurrentSoftware() Gets the current software in the system.
getCurrentSoftwareDescription() Gets the description of the current software.
getInstance() Gets the single instance of the Environment.
getSystem() Gets the system identifier.
getSystemDescription() Gets the system description.
setCurrentSoftware(int) Sets a new current software in the system.

Fields

- `private static Environment` **instance**
  - The unique instance of the environment.
- `public static final java.lang.String` **DEFAULT_NULL_STRING**
  - The default value to substitute when a null String is found.
- `public static final int` **DEFAULT_NULL_INT**
  - The default value to substitute when a null integer is found.

- `public static final double` **DEFAULT_NULL_DOUBLE**
  - The default value to substitute when a null double is found.
- `public static final java.sql.Date` **DEFAULT_NULL_DATE**
  - The default value to substitute when a null Date is found.
- `private java.lang.String` **systemID**
  - The identifier of the underlying system.
- `private java.lang.String` **systemDescription**
  - A short description of the underlying system.
- `private int` **currentSoftwareID**
  - The identifier of the current software version running in the system.
- `private java.lang.String` **currentSoftwareDescription**
  - A short description of the current software version.
- `private java.util.List` **softwareVersions**
  - Other software versions of the system.

Constructors

- **Environment**

```
private Environment()
```

  - **Description**

    Instantiates a new environment with the default values for the system and software.

- **Environment**

```
private Environment(java.lang.String systemID, java.lang.String
    systemDescription, int firstSoftwareVersion)
```

- **Description**

  Instantiates a new environment for a given system and first software version.

- **Parameters**

  * `systemID` – the identifier of the system
  * `systemDescription` – a short description of the system
  * `firstSoftwareVersion` – the identifier of the first software version to run in the system

Methods

- **getCurrentSoftware**

  ```
  public int getCurrentSoftware()
  ```

  - **Description**

    Gets the current software in the system.

  - **Returns** – the current software

- **getCurrentSoftwareDescription**

  ```
  public java.lang.String getCurrentSoftwareDescription()
  ```

  - **Description**

    Gets the description of the current software.

  - **Returns** – the description of the current software

- **getInstance**

  ```
  public static Environment getInstance()
  ```

- **Description**

  Gets the single instance of the Environment. If it still doesn't exist, creates a new instance with the default values.

- **Returns** – single instance of the Environment

- **getSystem**

```
public java.lang.String getSystem()
```

  - **Description**

    Gets the system identifier.

  - **Returns** – the system

- **getSystemDescription**

```
public java.lang.String getSystemDescription()
```

  - **Description**

    Gets the system description.

  - **Returns** – the system description

- **setCurrentSoftware**

```
public void setCurrentSoftware(int softwareID)
```

  - **Description**

    Sets a new current software in the system.

  - **Parameters**

    * `softwareID` – the identifier of the new current software

5 Package com.indra.iquality.dao

Package Contents *Page*

Interfaces

Classes

Defines the interfaces needed by the DAOs to retrieve and update data from a Data Base. Mantains flexibility by not coupling method signature of the data needed to the implementation of the data access or the physical storage of the data.

5.1 Interface BusinessCertificateDAO

The Interface to interact with the persistent representations of business certificates. See `BusinessCertificate` (in 1.2, page 111).

Declaration

```
public interface BusinessCertificateDAO
```

All known subinterfaces

BusinessCertificateDAOJDBCTemplateImpl (in 6.2, page 294), BusinessCertificateDAOJDBCTemplateImpl (in 6.2, page 294)

All classes known to implement interface

BusinessCertificateDAOJDBCTemplateImpl (in 6.2, page 294), BusinessCertificateDAOJDBCTemplateImpl (in 6.2, page 294)

Method summary

getAll(String, int) Gets all the records of business certifications for a given system and software version.
getCertificateConditions(String, int)
getCertificateDetails(String, String, int, String, int) Gets the details for an instance of `BusinessCertificate` (in 1.2, page 111) for a given system and software version.
getDetailHeaders(String, String, int) Gets the headers of the required fields to represent a detailed view of a `BusinessCertificate` (in 1.2, page 111) for a given system and software version.

Methods

- **getAll**

```
java.util.List getAll(java.lang.String sistema, int software)
```

- **Description**

  Gets all the records of business certifications for a given system and software version.

- **Parameters**

  * `sistema` – the current system

  * `software` – the current software version

- **Returns** – all the business certifications for the current system and software version

- **getCertificateConditions**

```
java.util.List getCertificateConditions(java.lang.String sistema
    ,int software)
```

- **getCertificateDetails**

```
java.util.List getCertificateDetails(java.lang.String idMes,java
    .lang.String idMetrica,int qttHeaders,java.lang.String
    sistema,int software)
```

- **Description**

  Gets the details for an instance of `BusinessCertificate` (in 1.2, page 111) for a given system and software version.

- **Parameters**

  * `idMes` – the month for which we want the details

  * `idMetrica` – the identifier for an instance of CertificacionDeNegocio

  * `qttHeaders` – the amount of fields the detailed view has; see `getDetailHeaders(String, String, int)`

  * `sistema` – the current system

  * `software` – the current software version

- **Returns** – a list of all the details of a business certification

- **getDetailHeaders**

```
java.util.List getDetailHeaders(java.lang.String idMetrica,java.
    lang.String sistema,int software)
```

  - **Description**

    Gets the headers of the required fields to represent a detailed view of a `BusinessCertificate` (in 1.2, page 111) for a given system and software version. Headers are dynamic, i.e., their content and amount may vary for different instances of the `BusinessCertificate` (in 1.2, page 111).

  - **Parameters**

    * `idMetrica` – the identifier for an instance of `BusinessCertificate` (in 1.2, page 111)
    * `sistema` – the current system
    * `software` – the current software version

  - **Returns** – a list of headers for the detailed view

5.2 Interface DependencyDAO

The Interface to interact with the persistent representations of job dependencies.

Declaration

```
public interface DependencyDAO
```

All known subinterfaces

DependencyDAOJDBCTemplateImpl (in 6.4, page 298), DependencyDAOJDBCTemplateImpl (in 6.4, page 298)

All classes known to implement interface

DependencyDAOJDBCTemplateImpl (in 6.4, page 298), DependencyDAOJDBCTemplateImpl (in 6.4, page 298)

Method summary

getAll(int, String, String, int) Gets the all the records of dependencies for a given job for a given system and software version.

Methods

- **getAll**

```
java.util.List getAll(int idEjecucion,java.lang.String idJob,
    java.lang.String sistema,int software)
```

  - **Description**

    Gets the all the records of dependencies for a given job for a given system and software version.

  - **Parameters**

    * `idEjecucion` – the identifier of the execution where the job takes part
    * `idJob` – the identifier of the job
    * `sistema` – the system
    * `software` – the software version

  - **Returns** – the all

5.3 Interface DictionaryOfConceptsDAO

The Interface to interact with the persistent representations of the dictionary of concepts, as well as the attributes and indicators that integrate it.

Declaration

```
public interface DictionaryOfConceptsDAO
```

All known subinterfaces

DictionaryOfConceptsDAOJDBCTemplateImpl (in 6.5, page 300), DictionaryOfConceptsDAOJDBCTemplateImpl (in 6.5, page 300)

All classes known to implement interface

DictionaryOfConceptsDAOJDBCTemplateImpl (in 6.5, page 300), DictionaryOfConceptsDAOJDBCTemplateImpl (in 6.5, page 300)

Method summary

getAllConcepts(String, int) Gets all the concepts of the dictionary for a given system and software version.
getDescriptionOfAttribute(String, String, String, int) Gets the description of an attribute component of the dictionary for a given system and software version.
getDescriptionOfIndicator(String, String, String, int) Gets the description of an indicator component of the dictionary for a given system and software version.
getDescriptionOfMasterAttribute(String, String, String, int) Gets the description of a master attribute component of the dictionary for a given system and software version.

Methods

- **getAllConcepts**

  ```
  java.util.List getAllConcepts(java.lang.String sistema, int
      software)
  ```

  – **Description**

  Gets all the concepts of the dictionary for a given system and software version.

- **Parameters**
  - `sistema` – the system
  - `software` – the software version
- **Returns** – all the concepts in the dictionary

- **getDescriptionOfAttribute**

```
com.indra.iquality.model.AttributeDescription
    getDescriptionOfAttribute(java.lang.String compRowID, java.
    lang.String ctRowID, java.lang.String sistema, int software)
```

- **Description**

  Gets the description of an attribute component of the dictionary for a given system and software version.

- **Parameters**
  - `compRowID` – the rowID of the component
  - `ctRowID` – the rowID of the ct
  - `sistema` – the system
  - `software` – the software version
- **Returns** – the description of the attribute

- **getDescriptionOfIndicator**

```
com.indra.iquality.model.IndicatorDescription
    getDescriptionOfIndicator(java.lang.String compRowID, java.
    lang.String ctRowID, java.lang.String sistema, int software)
```

- **Description**

  Gets the description of an indicator component of the dictionary for a given system and software version.

- **Parameters**

* `compRowID` – the rowID of the component
* `ctRowID` – the rowID of the ct
* `sistema` – the system
* `software` – the software version

– **Returns** – the description of the indicator

- **getDescriptionOfMasterAttribute**

```
com.indra.iquality.model.MasterAttributeDescription
    getDescriptionOfMasterAttribute(java.lang.String compRowID,
    java.lang.String ctRowID,java.lang.String sistema,int
    software)
```

– **Description**

Gets the description of a master attribute component of the dictionary for a given system and software version.

– **Parameters**

* `compRowID` – the rowID of the component
* `ctRowID` – the rowID of the ct
* `sistema` – the system
* `software` – the software version

– **Returns** – the description of the master attribute

5.4 Interface EnvironmentDAO

The Interface EnvironmentDAO.

Declaration

```
public interface EnvironmentDAO
```

All known subinterfaces

EnvironmentDAOJDBCTemplateImpl (in 6.7, page 307), EnvironmentDAOJDBCTemplateImpl (in 6.7, page 307)

All classes known to implement interface

EnvironmentDAOJDBCTemplateImpl (in 6.7, page 307), EnvironmentDAOJDBCTemplateImpl (in 6.7, page 307)

Method summary

getAllSystems() Gets all the systems available.
getAllTables(String) Gets the all tables defined in the environment system.
getCurrentSoftware(String) Gets the current software version of the environment system.

Methods

- **getAllSystems**

  ```
  java.util.List getAllSystems()
  ```

  - **Description**

    Gets all the systems available.

  - **Returns** – all systems in the form of pairs where the first value is the identifier of the system and the second value is the description of the system.

- **getAllTables**

  ```
  java.util.List getAllTables(java.lang.String sistema)
  ```

  - **Description**

    Gets the all tables defined in the environment system.

- **Parameters**
  - `sistema` – the system
- **Returns** – all the tables in the system, given in the form of pairs where the first value is the identifier of the table and the second value is the descriptive name of the table.

- **getCurrentSoftware**

```
org.apache.commons.lang3.tuple.Pair getCurrentSoftware(java.lang
    .String sistema)
```

  - **Description**

    Gets the current software version of the environment system.

  - **Parameters**
    - `sistema` – the system
  - **Returns** – a pair where the first value is the identifier of the software version and the second value is its description

5.5 Interface ExecutionDAO

The Interface to interact with the persistent representations of ETL executions.

Declaration

```
public interface ExecutionDAO
```

All known subinterfaces

ExecutionDAOJDBCTemplateImpl (in 6.9, page 311), ExecutionDAOJDBCTemplateImpl (in 6.9, page 311)

All classes known to implement interface

ExecutionDAOJDBCTemplateImpl (in 6.9, page 311), ExecutionDAOJDBCTemplateImpl (in 6.9, page 311)

Method summary

deleteById(int, String, int) Delete an execution with a given identifier.
getAll(String, int) Gets all the executions of a system with a software version.
getById(int, String, int) Gets an execution given its unique identifier for a system and software version.
save(Execution, String, int) Saves a persistent representation of an execution.
update(Execution, String, int) Update the persistent representation of an execution.

Methods

- **deleteById**

  ```
  boolean deleteById(int idEjecucion,java.lang.String sistema,int software)
  ```

  - **Description**

    Delete an execution with a given identifier.

  - **Parameters**

    * `idEjecucion` – the id of the execution to delete
    * `sistema` – the system
    * `software` – the software version

  - **Returns** – true, if successful

- **getAll**

  ```
  java.util.List getAll(java.lang.String sistema,int software)
    throws java.text.ParseException
  ```

- **Description**

  Gets all the executions of a system with a software version.

- **Parameters**

  * `sistema` – the system

  * `software` – the software version

- **Returns** – all the ETL executions

- **Throws**

  * `java.text.ParseException` – if a date is bad formatted

- **getById**

```
com.indra.iquality.model.Execution getById(int idEjecucion,java.
    lang.String sistema,int software)
```

  - **Description**

    Gets an execution given its unique identifier for a system and software version.

  - **Parameters**

    * `idEjecucion` – the identifier of the execution

    * `sistema` – the system

    * `software` – the software version

  - **Returns** – the execution

- **save**

```
boolean save(com.indra.iquality.model.Execution execution,java.
    lang.String sistema,int software)
```

  - **Description**

    Saves a persistent representation of an execution.

- **Parameters**
  - `execution` – the execution to save
  - `sistema` – the system
  - `software` – the software version
- **Returns** – true, if successful

- **update**

```
boolean update(com.indra.iquality.model.Execution execution,java
    .lang.String sistema,int software)
```

- **Description**

  Update the persistent representation of an execution.

- **Parameters**
  - `execution` – the new representation of the execution
  - `sistema` – the system
  - `software` – the software version
- **Returns** – true, if successful

5.6 Interface FlowDAO

The Interface to interact with the persistent representations of ETL flow definitions.

Declaration

```
public interface FlowDAO
```

All known subinterfaces

FlowDAOJDBCTemplateImpl (in 6.10, page 315), FlowDAOJDBCTemplateImpl (in 6.10, page 315)

All classes known to implement interface

FlowDAOJDBCTemplateImpl (in 6.10, page 315), FlowDAOJDBCTemplateImpl (in 6.10, page 315)

Method summary

getAll(String, int) Gets all the records of ETL flows stored in the underlying DB for the current version of the system.
save(Flow, String, int) Insert a new flow into the underlying DB for the current version of the system.

Methods

- **getAll**

```
java.util.List getAll(java.lang.String system, int software)
```

  - **Description**

    Gets all the records of ETL flows stored in the underlying DB for the current version of the system.

  - **Parameters**

    * `system` – the system

    * `software` – the software version

  - **Returns** – all the flows

- **save**

```
boolean save(com.indra.iquality.model.Flow flow, java.lang.String
    system, int software)
```

  - **Description**

    Insert a new flow into the underlying DB for the current version of the system. Idempotent method.

– **Parameters**

* `flow` – the flow to insert
* `system` – the system
* `software` – the software version

– **Returns** – true, if successful

5.7 Interface JobDAO

The Interface to interact with the persistent representations of atomic parts of an ETL executions, a.k.a, jobs.

Declaration

public interface JobDAO

All known subinterfaces

JobDAOJDBCTemplateImpl (in 6.11, page 317), JobDAOJDBCTemplateImpl (in 6.11, page 317)

All classes known to implement interface

JobDAOJDBCTemplateImpl (in 6.11, page 317), JobDAOJDBCTemplateImpl (in 6.11, page 317)

Method summary

deleteById(int, String, String, int) Delete a job with a given identifier and execution.
getAll(String, int) Gets all the jobs of in system with a software version.
getAllOfExecution(int, String, int) Gets all the jobs of an execution in a system with a software version.
getById(int, String, String, int) Gets an job given its unique identifier and its execution identifier for a system and software version.
update(Job, String, int) Update the persistent representation of a job.

Methods

- **deleteById**

  ```
  boolean deleteById(int idEjecucion,java.lang.String idJob,java.
      lang.String sistema,int software)
  ```

  - **Description**

    Delete a job with a given identifier and execution.

  - **Parameters**

    * `idEjecucion` – the identifier of the execution of the job
    * `idJob` – the identifier of the job
    * `sistema` – the system
    * `software` – the software version

  - **Returns** – true, if successful

- **getAll**

  ```
  java.util.List getAll(java.lang.String sistema,int software)
      throws java.text.ParseException
  ```

  - **Description**

    Gets all the jobs of in system with a software version.

  - **Parameters**

    * `sistema` – the system
    * `software` – the software version

  - **Returns** – all the jobs

  - **Throws**

    * `java.text.ParseException` – if a date is bad formatted

- **getAllOfExecution**

```
java.util.List getAllOfExecution(int idEjecucion,java.lang.
    String sistema,int software) throws java.text.ParseException
```

  - **Description**

    Gets all the jobs of an execution in a system with a software version.

  - **Parameters**

    * `idEjecucion` – the identifier of the execution of the job

    * `sistema` – the system

    * `software` – the software version

  - **Returns** – all the jobs of an execution

  - **Throws**

    * `java.text.ParseException` – if a date is bad formatted

- **getById**

```
com.indra.iquality.model.Job getById(int idEjecucion,java.lang.
    String idJob,java.lang.String sistema,int software)
```

  - **Description**

    Gets an job given its unique identifier and its execution identifier for a system and software version.

  - **Parameters**

    * `idEjecucion` – the identifier of the execution of the job

    * `idJob` – the identifier of the job

    * `sistema` – the system

    * `software` – the software version

- **Returns** – the job

- **update**

  ```
  boolean update(com.indra.iquality.model.Job job,java.lang.String
      sistema,int software)
  ```

  - **Description**

    Update the persistent representation of a job.

  - **Parameters**

    * `job` – the new representation of the job

    * `sistema` – the system

    * `software` – the software version

  - **Returns** – true, if successful

5.8 Interface RegisterOfOperationsDAO

The Interface to interact with the persistent representations of the register of operations of jobs.

Declaration

```
public interface RegisterOfOperationsDAO
```

All known subinterfaces

RegisterOfOperationsDAOJDBCTemplateImpl (in 6.12, page 321), RegisterOfOperationsDAOJDBCTemplateImpl (in 6.12, page 321)

All classes known to implement interface

RegisterOfOperationsDAOJDBCTemplateImpl (in 6.12, page 321), RegisterOfOperationsDAOJDBCTemplateImpl (in 6.12, page 321)

Method summary

getAll(int, String, String, int) Gets all the register of operations of a job from an execution, given a system with a software version.

Methods

- **getAll**

```
java.util.List getAll(int idEjecucion,java.lang.String idJob,
    java.lang.String sistema,int software) throws java.text.
    ParseException
```

  - **Description**

    Gets all the register of operations of a job from an execution, given a system with a software version.

  - **Parameters**

    * `idEjecucion` – the identifier of the execution of the job
    * `idJob` – the identifier of the job
    * `sistema` – the system
    * `software` – the software version

  - **Returns** – all the register of operations of a job

  - **Throws**

    * `java.text.ParseException` – if a date is bad formatted

5.9 Interface TechnicalCertificateDAO

The Interface to interact with the persistent representations of technical certificates.

Declaration

```
public interface TechnicalCertificateDAO
```

All known subinterfaces

TechnicalCertificateDAOJDBCTemplateImpl (in 6.13, page 323), TechnicalCertificateDAOJDBCTemplateImpl (in 6.13, page 323)

All classes known to implement interface

TechnicalCertificateDAOJDBCTemplateImpl (in 6.13, page 323), TechnicalCertificateDAOJDBCTemplateImpl (in 6.13, page 323)

Method summary

getAll(String, int) Gets all the technical certificates of a system with a software version.
getCertificateDetails(String, String, String, int) Gets the details of a technical certification given its metric and month of execution.
getHeaders() Gets the description of the headers of the detailed view of the last certificate consulted.
getLastNumCols() Gets the number of dimensions of the detailed view of the last certificate consulted.
getValidationConditions(String, int)

Methods

- **getAll**

  ```
  java.util.List getAll(java.lang.String sistema, int software)
  ```

  - **Description**

    Gets all the technical certificates of a system with a software version.

  - **Parameters**

    * `sistema` – the system

    * `software` – the software version

  – **Returns** – all the technical certificates

- **getCertificateDetails**

```
java.util.List getCertificateDetails(java.lang.String idMetrica,
    java.lang.String idMes,java.lang.String sistema,int software)
```

  – **Description**

    Gets the details of a technical certification given its metric and month of execution. Updates the values of the headers and number of columns of the detailed view. @see #getLastNumCols() and @see #getHeaders().

  – **Parameters**

    * `idMetrica` – the identifier of the metric of the certificate

    * `idMes` – the month of execution of the certificate

    * `sistema` – the system

    * `software` – the software version

  – **Returns** – the details of a technical certification

- **getHeaders**

```
java.util.List getHeaders()
```

  – **Description**

    Gets the description of the headers of the detailed view of the last certificate consulted. In other words, the text to display in the columns that the detailed view should have after the last call to `getCertificateDetails(String, String, String, int)` . So this method SHOULD NOT be called before `getCertificateDetails(String, String, String, int)` .

  – **Returns** – the headers of the columns of the detailed view

- **getLastNumCols**

  ```
  int getLastNumCols()
  ```

  - **Description**

    Gets the number of dimensions of the detailed view of the last certificate consulted. In other words, the number of columns that the detailed view should have after the last call to `getCertificateDetails(String, String, String, int)` . So this method SHOULD NOT be called before `getCertificateDetails(String, String, String, int)` .

  - **Returns** – the number of columns of the detailed view

- **getValidationConditions**

  ```
  java.util.List getValidationConditions(java.lang.String sistema,
      int software)
  ```

5.10 Interface TraceOfRegisterDAO

The Interface to interact with the persistent representations of register traces of operations.

Declaration

```
public interface TraceOfRegisterDAO
```

All known subinterfaces

TraceOfRegisterDAOJDBCTemplateImpl (in 6.14, page 327), TraceOfRegisterDAOJDBCTemplateImpl (in 6.14, page 327)

All classes known to implement interface

TraceOfRegisterDAOJDBCTemplateImpl (in 6.14, page 327), TraceOfRegisterDAOJDBCTemplateImpl (in 6.14, page 327)

Method summary

getAll(int, String, int) Gets all the register traces of an operation for a given system and software version.

Methods

- **getAll**

```
java.util.List getAll(int idOperacion,java.lang.String sistema,
    int software) throws java.lang.Exception
```

  - **Description**

    Gets all the register traces of an operation for a given system and software version.

  - **Parameters**

    * `idOperacion` – the identifier of the operation
    * `sistema` – the system
    * `software` – the software version

  - **Returns** – the all

  - **Throws**

    * `java.lang.Exception` – all the register traces of the operation

5.11 Class JExcelTest

Declaration

```
public class JExcelTest
 extends java.lang.Object
```

Constructor summary

JExcelTest()

Method summary

testJExcelAPI()

Constructors

- **JExcelTest**

```
public JExcelTest()
```

Methods

- **testJExcelAPI**

```
public void testJExcelAPI()
```

6 Package com.indra.iquality.dao.jdbctemplateimplem

Package Contents *Page*

Classes

Provides the implementation of the DAO interfaces connecting to an Oracle DB. Uses the Spring JDBCTemplate helper to ease the mapping and querying.

6.1 Class AbstractDAOJDBCTemplateImpl

The Abstract Class AbstractDAOJDBCTemplateImpl. Encapsulates the common parts of all the JDCBTemplate implementations of the DAO to keep the code DRY.

Declaration

```
public abstract class AbstractDAOJDBCTemplateImpl
 extends java.lang.Object
```

All known subclasses

FlowDAOJDBCTemplateImpl (in 6.10, page 315), BusinessCertificateDAOJDBCTemplateImpl (in 6.2, page 294), JobDAOJDBCTemplateImpl (in 6.11, page 317), DependencyDAOJDBCTemplateImpl

(in 6.4, page 298), TechnicalCertificateDAOJDBCTemplateImpl (in 6.13, page 323), EnvironmentDAOJDBCTemplateImpl (in 6.7, page 307), ExecutionDAOJDBCTemplateImpl (in 6.9, page 311), TraceOfRegisterDAOJDBCTemplateImpl (in 6.14, page 327), DictionaryOfConceptsDAOJDBCTemplateImpl (in 6.5, page 300), RegisterOfOperationsDAOJDBCTemplateImpl (in 6.12, page 321), TraceOfRegisterDAOJDBCTemplateImpl (in 6.14, page 327), TechnicalCertificateDAOJDBCTemplateImpl (in 6.13, page 323), RegisterOfOperationsDAOJDBCTemplateImpl (in 6.12, page 321), JobDAOJDBCTemplateImpl (in 6.11, page 317), FlowDAOJDBCTemplateImpl (in 6.10, page 315), ExecutionDAOJDBCTemplateImpl (in 6.9, page 311), EnvironmentDAOJDBCTemplateImpl (in 6.7, page 307), DictionaryOfConceptsDAOJDBCTemplateImpl (in 6.5, page 300), DependencyDAOJDBCTemplateImpl (in 6.4, page 298), BusinessCertificateDAOJDBCTemplateImpl (in 6.2, page 294)

Field summary

dataSource The source of the data (the DB).
helper The helper with common utilities.

Constructor summary

AbstractDAOJDBCTemplateImpl()

Method summary

setDataSource(DataSource) Sets the data source of the DAO.

Fields

- `protected javax.sql.DataSource` **dataSource**
  - The source of the data (the DB).
- `protected com.indra.iquality.helper.DataHelper` **helper**
  - The helper with common utilities.

Constructors

- **AbstractDAOJDBCTemplateImpl**

```
public AbstractDAOJDBCTemplateImpl()
```

Methods

- **setDataSource**

  ```
  public void setDataSource(javax.sql.DataSource dataSource)
  ```

  - **Description**

    Sets the data source of the DAO.

  - **Parameters**

    * `dataSource` – the new data source

6.2 Class BusinessCertificateDAOJDBCTemplateImpl

Implementation of `BusinessCertificateDAO` (in 5.1, page 269) using JDBC to connect to an Oracle DB.

Declaration

```
public class BusinessCertificateDAOJDBCTemplateImpl
 extends com.indra.iquality.dao.jdbctemplateimplem.
    AbstractDAOJDBCTemplateImpl implements com.indra.iquality.dao.
    BusinessCertificateDAO
```

Field summary

logger The Constant logger.

Constructor summary

BusinessCertificateDAOJDBCTemplateImpl()

Method summary

getAll(String, int)
getCertificateConditions(String, int)
getCertificateDetails(String, String, int, String, int)
getDetailHeaders(String, String, int)

Fields

- `private static final org.slf4j.Logger` **logger**
  - The Constant logger.

Constructors

- **BusinessCertificateDAOJDBCTemplateImpl**

  ```
  public BusinessCertificateDAOJDBCTemplateImpl()
  ```

Methods

- **getAll**

  ```
  java.util.List getAll(java.lang.String sistema, int software)
  ```

  - **Description copied from com.indra.iquality.dao.BusinessCertificateDAO** (**in 5.1, page 269**)

    Gets all the records of business certifications for a given system and software version.

  - **Parameters**
    - `sistema` – the current system
    - `software` – the current software version
  - **Returns** – all the business certifications for the current system and software version

- **getCertificateConditions**

```
java.util.List getCertificateConditions(java.lang.String sistema
    ,int software)
```

- **getCertificateDetails**

```
java.util.List getCertificateDetails(java.lang.String idMes,java
    .lang.String idMetrica,int qttHeaders,java.lang.String
    sistema,int software)
```

  - **Description copied from com.indra.iquality.dao.BusinessCertificateDAO** (**in 5.1, page 269**)

    Gets the details for an instance of `BusinessCertificate` (in 1.2, page 111) for a given system and software version.

  - **Parameters**

    * `idMes` – the month for which we want the details
    * `idMetrica` – the identifier for an instance of CertificacionDeNegocio
    * `qttHeaders` – the amount of fields the detailed view has; see `getDetailHeaders(String, String, int)`
    * `sistema` – the current system
    * `software` – the current software version

  - **Returns** – a list of all the details of a business certification

- **getDetailHeaders**

```
java.util.List getDetailHeaders(java.lang.String idMetrica,java.
    lang.String sistema,int software)
```

  - **Description copied from com.indra.iquality.dao.BusinessCertificateDAO** (**in 5.1, page 269**)

    Gets the headers of the required fields to represent a detailed view of a `BusinessCertificate` (in 1.2, page 111) for a given system and software version. Headers are dynamic, i.e.,

their content and amount may vary for different instances of the `BusinessCertificate` (in 1.2, page 111).

– **Parameters**

* `idMetrica` – the identifier for an instance of `BusinessCertificate` (in 1.2, page 111)

* `sistema` – the current system

* `software` – the current software version

– **Returns** – a list of headers for the detailed view

Members inherited from class AbstractDAOJDBCTemplateImpl

`com.indra.iquality.dao.jdbctemplateimplem.AbstractDAOJDBCTemplateImpl` (in 6.1, page 292)

- `protected` **dataSource**
- `protected` **helper**
- `public void` **setDataSource**`(javax.sql.DataSource` **dataSource**`)`

6.3 Class CertificacionDeNegocioDAOJDBCTemplateImplTest

Declaration

public class CertificacionDeNegocioDAOJDBCTemplateImplTest
extends java.lang.Object

Field summary

environment

Constructor summary

CertificacionDeNegocioDAOJDBCTemplateImplTest()

Method summary

testGetAll()
testGetDetallesDeCertificacion()
testGetHeadersDetalles()

Fields

- `private static final com.indra.iquality.singleton.Environment` **environment**

Constructors

- **CertificacionDeNegocioDAOJDBCTemplateImplTest**

  public CertificacionDeNegocioDAOJDBCTemplateImplTest()

Methods

- **testGetAll**

  public void testGetAll()

- **testGetDetallesDeCertificacion**

  public void testGetDetallesDeCertificacion()

- **testGetHeadersDetalles**

  public void testGetHeadersDetalles()

6.4 Class DependencyDAOJDBCTemplateImpl

Implementation of `DependencyDAO` (in 5.2, page 271) using JDBC to connect to an Oracle DB.

Declaration

```
public class DependencyDAOJDBCTemplateImpl
 extends com.indra.iquality.dao.jdbctemplateimplem.
    AbstractDAOJDBCTemplateImpl implements com.indra.iquality.dao.
    DependencyDAO
```

Field summary

logger The Constant logger.

Constructor summary

DependencyDAOJDBCTemplateImpl()

Method summary

getAll(int, String, String, int)

Fields

- `private static final org.slf4j.Logger` **logger**
  - The Constant logger.

Constructors

- **DependencyDAOJDBCTemplateImpl**

  public DependencyDAOJDBCTemplateImpl()

Methods

- **getAll**

```
java.util.List getAll(int idEjecucion,java.lang.String idJob,
    java.lang.String sistema,int software)
```

– **Description copied from com.indra.iquality.dao.DependencyDAO** (**in 5.2, page 271**)

Gets the all the records of dependencies for a given job for a given system and software version.

– **Parameters**

* `idEjecucion` – the identifier of the execution where the job takes part
* `idJob` – the identifier of the job
* `sistema` – the system
* `software` – the software version

– **Returns** – the all

Members inherited from class AbstractDAOJDBCTemplateImpl

`com.indra.iquality.dao.jdbctemplateimplem.AbstractDAOJDBCTemplateImpl` (in 6.1, page 292)

- `protected` **dataSource**
- `protected` **helper**
- `public void` **setDataSource**`(javax.sql.DataSource` **dataSource**`)`

6.5 Class DictionaryOfConceptsDAOJDBCTemplateImpl

Implementation of `DictionaryOfConceptsDAO` (in 5.3, page 272) using JDBC to connect to an Oracle DB.

Declaration

```
public class DictionaryOfConceptsDAOJDBCTemplateImpl
```

extends com.indra.iquality.dao.jdbctemplateimplem.AbstractDAOJDBCTemplateImpl **implements** com.indra.iquality.dao.DictionaryOfConceptsDAO

Field summary

logger The Constant logger.

Constructor summary

DictionaryOfConceptsDAOJDBCTemplateImpl()

Method summary

getAllConcepts(String, int)
getCertifications(int, String, int) Gets the certifications of a component for a given system and software version.
getDescriptionOfAttribute(String, String, String, int)
getDescriptionOfIndicator(String, String, String, int)
getDescriptionOfMasterAttribute(String, String, String, int)

Fields

- `private static final org.slf4j.Logger` **logger**
  - The Constant logger.

Constructors

- **DictionaryOfConceptsDAOJDBCTemplateImpl**

  public DictionaryOfConceptsDAOJDBCTemplateImpl()

Methods

- **getAllConcepts**

```
java.util.List getAllConcepts(java.lang.String sistema, int
    software)
```

  - **Description copied from com.indra.iquality.dao.DictionaryOfConceptsDAO** (**in 5.3, page 272**)

    Gets all the concepts of the dictionary for a given system and software version.

  - **Parameters**
    - `sistema` – the system
    - `software` – the software version
  - **Returns** – all the concepts in the dictionary

- **getCertifications**

```
private java.util.List getCertifications(int idComponente, java.
    lang.String sistema, int software)
```

  - **Description**

    Gets the certifications of a component for a given system and software version.

  - **Parameters**
    - `idComponente` – the identifier of the component
    - `sistema` – the system
    - `software` – the software version
  - **Returns** – the certifications of the component

- **getDescriptionOfAttribute**

```
com.indra.iquality.model.AttributeDescription
    getDescriptionOfAttribute(java.lang.String compRowID, java.
    lang.String ctRowID, java.lang.String sistema, int software)
```

- **Description copied from com.indra.iquality.dao.DictionaryOfConceptsDAO** (**in 5.3, page 272**)

  Gets the description of an attribute component of the dictionary for a given system and software version.

- **Parameters**
  - `compRowID` – the rowID of the component
  - `ctRowID` – the rowID of the ct
  - `sistema` – the system
  - `software` – the software version
- **Returns** – the description of the attribute

- **getDescriptionOfIndicator**

```
com.indra.iquality.model.IndicatorDescription
    getDescriptionOfIndicator(java.lang.String compRowID, java.
    lang.String ctRowID, java.lang.String sistema, int software)
```

- **Description copied from com.indra.iquality.dao.DictionaryOfConceptsDAO** (**in 5.3, page 272**)

  Gets the description of an indicator component of the dictionary for a given system and software version.

- **Parameters**
  - `compRowID` – the rowID of the component
  - `ctRowID` – the rowID of the ct
  - `sistema` – the system
  - `software` – the software version

– **Returns** – the description of the indicator

- **getDescriptionOfMasterAttribute**

```
com.indra.iquality.model.MasterAttributeDescription
    getDescriptionOfMasterAttribute(java.lang.String compRowID,
    java.lang.String ctRowID,java.lang.String sistema,int
    software)
```

– **Description copied from com.indra.iquality.dao.DictionaryOfConceptsDAO** (**in 5.3, page 272**)

Gets the description of a master attribute component of the dictionary for a given system and software version.

– **Parameters**

* `compRowID` – the rowID of the component
* `ctRowID` – the rowID of the ct
* `sistema` – the system
* `software` – the software version

– **Returns** – the description of the master attribute

Members inherited from class AbstractDAOJDBCTemplateImpl

`com.indra.iquality.dao.jdbctemplateimplem.AbstractDAOJDBCTemplateImpl` (in 6.1, page 292)

- `protected` **dataSource**
- `protected` **helper**
- `public void` **setDataSource**`(javax.sql.DataSource` **dataSource**`)`

6.6 Class DictionaryOfConceptsDAOJDBCTemplateImpl.Certification

The auxiliary Class for representing certifications.

Declaration

```
private class DictionaryOfConceptsDAOJDBCTemplateImpl.Certification
 extends java.lang.Object
```

Field summary

descripcion The description of the component.
idComponente The identifier of the component.
nombre The name of the component.

Constructor summary

Certification()

Method summary

getDescripcion() Gets description of the component.
getIdComponente() Gets the identifier of the component.
getNombre() Gets name of the component.
setDescripcion(String) Sets description of the component.
setIdComponente(int) Sets the identifier of the component.
setNombre(String) Sets name of the component

Fields

- `private int` **idComponente**
  - The identifier of the component.
- `private java.lang.String` **nombre**
  - The name of the component.
- `private java.lang.String` **descripcion**
  - The description of the component.

Constructors

- **Certification**

  ```
  private Certification()
  ```

Methods

- **getDescripcion**

  ```
  public java.lang.String getDescripcion()
  ```

  - **Description**

    Gets description of the component.

  - **Returns** – description of the component

- **getIdComponente**

  ```
  public int getIdComponente()
  ```

  - **Description**

    Gets the identifier of the component.

  - **Returns** – the identifier of the component

- **getNombre**

  ```
  public java.lang.String getNombre()
  ```

  - **Description**

    Gets name of the component.

  - **Returns** – name of the component

- **setDescripcion**

  ```
  public void setDescripcion(java.lang.String descripcion)
  ```

  - **Description**

    Sets description of the component.

  - **Parameters**

    * `descripcion` – the new description of the component

- **setIdComponente**

  ```
  public void setIdComponente(int idComponente)
  ```

  - **Description**

    Sets the identifier of the component.

  - **Parameters**

    * `idComponente` – the new identifier of the component

- **setNombre**

  ```
  public void setNombre(java.lang.String nombre)
  ```

  - **Description**

    Sets name of the component

  - **Parameters**

    * `nombre` – the new name of the component

6.7 Class EnvironmentDAOJDBCTemplateImpl

The Class EnvironmentDAOJDBCTemplateImpl.

Declaration

```
public class EnvironmentDAOJDBCTemplateImpl
 extends com.indra.iquality.dao.jdbctemplateimplem.
    AbstractDAOJDBCTemplateImpl implements com.indra.iquality.dao.
    EnvironmentDAO
```

Field summary

logger The Constant logger.

Constructor summary

EnvironmentDAOJDBCTemplateImpl()

Method summary

getAllSystems()
getAllTables(String)
getCurrentSoftware(String)

Fields

- `private static final org.slf4j.Logger` **logger**
  - The Constant logger.

Constructors

- **EnvironmentDAOJDBCTemplateImpl**

  public EnvironmentDAOJDBCTemplateImpl()

Methods

- **getAllSystems**

  ```
  java.util.List getAllSystems()
  ```

  - **Description copied from com.indra.iquality.dao.EnvironmentDAO** (in 5.4, page 275)

    Gets all the systems available.

  - **Returns** – all systems in the form of pairs where the first value is the identifier of the system and the second value is the description of the system.

- **getAllTables**

  ```
  java.util.List getAllTables(java.lang.String sistema)
  ```

  - **Description copied from com.indra.iquality.dao.EnvironmentDAO** (in 5.4, page 275)

    Gets the all tables defined in the environment system.

  - **Parameters**

    * `sistema` – the system

  - **Returns** – all the tables in the system, given in the form of pairs where the first value is the identifier of the table and the second value is the descriptive name of the table.

- **getCurrentSoftware**

  ```
  org.apache.commons.lang3.tuple.Pair getCurrentSoftware(java.lang
      .String sistema)
  ```

  - **Description copied from com.indra.iquality.dao.EnvironmentDAO** (in 5.4, page 275)

    Gets the current software version of the environment system.

– **Parameters**

* `sistema` – the system

– **Returns** – a pair where the first value is the identifier of the software version and the second value is its description

Members inherited from class AbstractDAOJDBCTemplateImpl

`com.indra.iquality.dao.jdbctemplateimplem.AbstractDAOJDBCTemplateImpl` (in 6.1, page 292)

- `protected` **dataSource**
- `protected` **helper**
- `public void` **setDataSource**`(javax.sql.DataSource` **dataSource**`)`

6.8 Class EnvironmentDAOJDBCTemplateImplTest

Declaration

```
public class EnvironmentDAOJDBCTemplateImplTest
 extends java.lang.Object
```

Field summary

environment

Constructor summary

EnvironmentDAOJDBCTemplateImplTest()

Method summary

testGetCurrentSoftware()

Fields

- `private static final com.indra.iquality.singleton.Environment` **environment**

Constructors

- **EnvironmentDAOJDBCTemplateImplTest**

  public EnvironmentDAOJDBCTemplateImplTest()

Methods

- **testGetCurrentSoftware**

  public void testGetCurrentSoftware()

6.9 Class ExecutionDAOJDBCTemplateImpl

Implementation of `ExecutionDAO` (in 5.5, page 277) using JDBC to connect to an Oracle DB.

Declaration

public class ExecutionDAOJDBCTemplateImpl
extends com.indra.iquality.dao.jdbctemplateimplem.
AbstractDAOJDBCTemplateImpl **implements** com.indra.iquality.dao.
ExecutionDAO

Field summary

logger The Constant logger.

Constructor summary

ExecutionDAOJDBCTemplateImpl()

Method summary

deleteById(int, String, int)
getAll(String, int)
getById(int, String, int)
save(Execution, String, int)
update(Execution, String, int)

Fields

- `private static final org.slf4j.Logger` **logger**
  - The Constant logger.

Constructors

- **ExecutionDAOJDBCTemplateImpl**

  public ExecutionDAOJDBCTemplateImpl()

Methods

- **deleteById**

  boolean deleteById(**int** idEjecucion, java.lang.String sistema, **int** software)

  - **Description copied from com.indra.iquality.dao.ExecutionDAO** (in 5.5, page 277)

    Delete an execution with a given identifier.

  - **Parameters**
    - `idEjecucion` – the id of the execution to delete
    - `sistema` – the system

* `software` – the software version

– **Returns** – true, if successful

- **getAll**

```
java.util.List getAll(java.lang.String sistema, int software)
    throws java.text.ParseException
```

– **Description copied from com.indra.iquality.dao.ExecutionDAO** (**in 5.5, page 277**)

Gets all the executions of a system with a software version.

– **Parameters**

* `sistema` – the system

* `software` – the software version

– **Returns** – all the ETL executions

– **Throws**

* `java.text.ParseException` – if a date is bad formatted

- **getById**

```
com.indra.iquality.model.Execution getById(int idEjecucion, java.
    lang.String sistema, int software)
```

– **Description copied from com.indra.iquality.dao.ExecutionDAO** (**in 5.5, page 277**)

Gets an execution given its unique identifier for a system and software version.

– **Parameters**

* `idEjecucion` – the identifier of the execution

* `sistema` – the system

* `software` – the software version

– **Returns** – the execution

- **save**

  boolean save(com.indra.iquality.model.Execution execution,java.lang.String sistema,**int** software)

  – **Description copied from com.indra.iquality.dao.ExecutionDAO** (**in 5.5, page 277**)

    Saves a persistent representation of an execution.

  – **Parameters**

    * `execution` – the execution to save

    * `sistema` – the system

    * `software` – the software version

  – **Returns** – true, if successful

- **update**

  boolean update(com.indra.iquality.model.Execution execution,java.lang.String sistema,**int** software)

  – **Description copied from com.indra.iquality.dao.ExecutionDAO** (**in 5.5, page 277**)

    Update the persistent representation of an execution.

  – **Parameters**

    * `execution` – the new representation of the execution

    * `sistema` – the system

    * `software` – the software version

  – **Returns** – true, if successful

Members inherited from class AbstractDAOJDBCTemplateImpl

`com.indra.iquality.dao.jdbctemplateimplem.AbstractDAOJDBCTemplateImpl` (in 6.1, page 292)

- `protected` **dataSource**
- `protected` **helper**
- `public void` **setDataSource**`(javax.sql.DataSource` **dataSource**`)`

6.10 Class FlowDAOJDBCTemplateImpl

The Class FlowDAOJDBCTemplateImpl.

Declaration

public class FlowDAOJDBCTemplateImpl
extends com.indra.iquality.dao.jdbctemplateimplem.AbstractDAOJDBCTemplateImpl **implements** com.indra.iquality.dao.FlowDAO

Field summary

logger The Constant logger.

Constructor summary

FlowDAOJDBCTemplateImpl()

Method summary

getAll(String, int)
save(Flow, String, int)

Fields

- `private static final org.slf4j.Logger` **logger**

– The Constant logger.

Constructors

- **FlowDAOJDBCTemplateImpl**

  ```
  public FlowDAOJDBCTemplateImpl()
  ```

Methods

- **getAll**

  ```
  java.util.List getAll(java.lang.String system, int software)
  ```

  – **Description copied from com.indra.iquality.dao.FlowDAO** (in 5.6, page 280)

  Gets all the records of ETL flows stored in the underlying DB for the current version of the system.

  – **Parameters**

  * `system` – the system

  * `software` – the software version

  – **Returns** – all the flows

- **save**

  ```
  boolean save(com.indra.iquality.model.Flow flow, java.lang.String
      system, int software)
  ```

  – **Description copied from com.indra.iquality.dao.FlowDAO** (in 5.6, page 280)

  Insert a new flow into the underlying DB for the current version of the system. Idempotent method.

  – **Parameters**

* `flow` – the flow to insert
* `system` – the system
* `software` – the software version

– **Returns** – true, if successful

Members inherited from class AbstractDAOJDBCTemplateImpl

`com.indra.iquality.dao.jdbctemplateimplem.AbstractDAOJDBCTemplateImpl` (in 6.1, page 292)

- `protected` **dataSource**
- `protected` **helper**
- `public void` **setDataSource**`(javax.sql.DataSource` **dataSource**`)`

6.11 Class JobDAOJDBCTemplateImpl

Implementation of `JobDAO` (in 5.7, page 282) using JDBC to connect to an Oracle DB.

Declaration

```
public class JobDAOJDBCTemplateImpl
 extends com.indra.iquality.dao.jdbctemplateimplem.
    AbstractDAOJDBCTemplateImpl implements com.indra.iquality.dao.
    JobDAO
```

Field summary

logger The Constant logger.

Constructor summary

JobDAOJDBCTemplateImpl()

Method summary

deleteById(int, String, String, int)
getAll(String, int)
getAllOfExecution(int, String, int)
getById(int, String, String, int)
update(Job, String, int)

Fields

- `private static final org.slf4j.Logger` **logger**
  - The Constant logger.

Constructors

- **JobDAOJDBCTemplateImpl**

  public JobDAOJDBCTemplateImpl()

Methods

- **deleteById**

  boolean deleteById(**int** idEjecucion, java.lang.String idJob, java.lang.String sistema, **int** software)

  - **Description copied from com.indra.iquality.dao.JobDAO** (in 5.7, page 282)

    Delete a job with a given identifier and execution.

  - **Parameters**
    - `idEjecucion` – the identifier of the execution of the job
    - `idJob` – the identifier of the job
    - `sistema` – the system

* `software` – the software version

– **Returns** – true, if successful

- **getAll**

```
java.util.List getAll(java.lang.String sistema,int software)
    throws java.text.ParseException
```

– **Description copied from com.indra.iquality.dao.JobDAO** (in 5.7, page 282)

Gets all the jobs of in system with a software version.

– **Parameters**

* `sistema` – the system

* `software` – the software version

– **Returns** – all the jobs

– **Throws**

* `java.text.ParseException` – if a date is bad formatted

- **getAllOfExecution**

```
java.util.List getAllOfExecution(int idEjecucion,java.lang.
    String sistema,int software) throws java.text.ParseException
```

– **Description copied from com.indra.iquality.dao.JobDAO** (in 5.7, page 282)

Gets all the jobs of an execution in a system with a software version.

– **Parameters**

* `idEjecucion` – the identifier of the execution of the job

* `sistema` – the system

* `software` – the software version

– **Returns** – all the jobs of an execution

- **Throws**
  - `java.text.ParseException` – if a date is bad formatted

- **getById**

```
com.indra.iquality.model.Job getById(int idEjecucion,java.lang.
    String idJob,java.lang.String sistema,int software)
```

  - **Description copied from com.indra.iquality.dao.JobDAO** (in 5.7, page 282)

    Gets an job given its unique identifier and its execution identifier for a system and software version.

  - **Parameters**
    - `idEjecucion` – the identifier of the execution of the job
    - `idJob` – the identifier of the job
    - `sistema` – the system
    - `software` – the software version
  - **Returns** – the job

- **update**

```
boolean update(com.indra.iquality.model.Job job,java.lang.String
    sistema,int software)
```

  - **Description copied from com.indra.iquality.dao.JobDAO** (in 5.7, page 282)

    Update the persistent representation of a job.

  - **Parameters**
    - `job` – the new representation of the job
    - `sistema` – the system
    - `software` – the software version

– **Returns** – true, if successful

Members inherited from class AbstractDAOJDBCTemplateImpl

`com.indra.iquality.dao.jdbctemplateimplem.AbstractDAOJDBCTemplateImpl` (in 6.1, page 292)

- `protected` **dataSource**
- `protected` **helper**
- `public void` **setDataSource**`(javax.sql.DataSource` **dataSource**`)`

6.12 Class RegisterOfOperationsDAOJDBCTemplateImpl

Implementation of `RegisterOfOperationsDAO` (in 5.8, page 285) using JDBC to connect to an Oracle DB.

Declaration

```
public class RegisterOfOperationsDAOJDBCTemplateImpl
 extends com.indra.iquality.dao.jdbctemplateimplem.
    AbstractDAOJDBCTemplateImpl implements com.indra.iquality.dao.
    RegisterOfOperationsDAO
```

Field summary

logger The Constant logger.

Constructor summary

RegisterOfOperationsDAOJDBCTemplateImpl()

Method summary

getAll(int, String, String, int)

Fields

- `private static final org.slf4j.Logger` **logger**
  - The Constant logger.

Constructors

- **RegisterOfOperationsDAOJDBCTemplateImpl**

  ```
  public RegisterOfOperationsDAOJDBCTemplateImpl()
  ```

Methods

- **getAll**

  ```
  java.util.List getAll(int idEjecucion,java.lang.String idJob,
      java.lang.String sistema,int software) throws java.text.
      ParseException
  ```

  - **Description copied from com.indra.iquality.dao.RegisterOfOperationsDAO** (**in 5.8, page 285**)

    Gets all the register of operations of a job from an execution, given a system with a software version.

  - **Parameters**
    - `idEjecucion` – the identifier of the execution of the job
    - `idJob` – the identifier of the job
    - `sistema` – the system
    - `software` – the software version
  - **Returns** – all the register of operations of a job
  - **Throws**

* `java.text.ParseException` – if a date is bad formatted

Members inherited from class AbstractDAOJDBCTemplateImpl

`com.indra.iquality.dao.jdbctemplateimplem.AbstractDAOJDBCTemplateImpl` (in 6.1, page 292)

- `protected` **dataSource**
- `protected` **helper**
- `public void` **setDataSource**`(javax.sql.DataSource` **dataSource**`)`

6.13 Class TechnicalCertificateDAOJDBCTemplateImpl

Implementation of `TechnicalCertificateDAO` (in 5.9, page 286) using JDBC to connect to an Oracle DB.

Declaration

```
public class TechnicalCertificateDAOJDBCTemplateImpl
 extends com.indra.iquality.dao.jdbctemplateimplem.
    AbstractDAOJDBCTemplateImpl implements com.indra.iquality.dao.
    TechnicalCertificateDAO
```

Field summary

headers The description of the headers of the detailed view of the last certificate consulted.
lastNumCols The description of the headers of the detailed view of the last certificate consulted.
logger The Constant logger.

Constructor summary

TechnicalCertificateDAOJDBCTemplateImpl()

Method summary

getAll(String, int)
getCertificateDetails(String, String, String, int)
getHeaders()
getLastNumCols()
getQuery(String, String, String, int) Auxiliary method to get the query needed to get the details of a technical certificate.
getValidationConditions(String, int)

Fields

- `private static final org.slf4j.Logger` **logger**
  - The Constant logger.
- `private int` **lastNumCols**
  - The description of the headers of the detailed view of the last certificate consulted.
- `private java.util.List` **headers**
  - The description of the headers of the detailed view of the last certificate consulted.

Constructors

- **TechnicalCertificateDAOJDBCTemplateImpl**

  ```
  public TechnicalCertificateDAOJDBCTemplateImpl()
  ```

Methods

- **getAll**

  ```
  java.util.List getAll(java.lang.String sistema, int software)
  ```

  - **Description copied from com.indra.iquality.dao.TechnicalCertificateDAO** (**in 5.9, page 286**)

Gets all the technical certificates of a system with a software version.

– **Parameters**

* `sistema` – the system

* `software` – the software version

– **Returns** – all the technical certificates

- **getCertificateDetails**

```
java.util.List getCertificateDetails(java.lang.String idMetrica,
    java.lang.String idMes,java.lang.String sistema,int software)
```

– **Description copied from com.indra.iquality.dao.TechnicalCertificateDAO** (**in 5.9, page 286**)

Gets the details of a technical certification given its metric and month of execution. Updates the values of the headers and number of columns of the detailed view. @see #getLastNumCols() and @see #getHeaders().

– **Parameters**

* `idMetrica` – the identifier of the metric of the certificate

* `idMes` – the month of execution of the certificate

* `sistema` – the system

* `software` – the software version

– **Returns** – the details of a technical certification

- **getHeaders**

```
java.util.List getHeaders()
```

– **Description copied from com.indra.iquality.dao.TechnicalCertificateDAO** (**in 5.9, page 286**)

Gets the description of the headers of the detailed view of the last certificate consulted. In other words, the text to display in the columns that the detailed view should

have after the last call to `getCertificateDetails(String, String, String, int)` . So this method SHOULD NOT be called before `getCertificateDetails(String, String, String, int)` .

– **Returns** – the headers of the columns of the detailed view

- **getLastNumCols**

```
int getLastNumCols()
```

– **Description copied from com.indra.iquality.dao.TechnicalCertificateDAO** (**in 5.9, page 286**)

Gets the number of dimensions of the detailed view of the last certificate consulted. In other words, the number of columns that the detailed view should have after the last call to `getCertificateDetails(String, String, String, int)` . So this method SHOULD NOT be called before `getCertificateDetails(String, String, String, int)` .

– **Returns** – the number of columns of the detailed view

- **getQuery**

```
private java.lang.String getQuery(java.lang.String idMetrica,
    java.lang.String idMes,java.lang.String sistema,int software)
```

– **Description**

Auxiliary method to get the query needed to get the details of a technical certificate.

– **Parameters**

* `idMetrica` – the identifier of the metric of the certificate
* `idMes` – the month of execution of the certificate
* `sistema` – the system
* `software` – the software version

– **Returns** – the query to get the details

- **getValidationConditions**

```
java.util.List getValidationConditions(java.lang.String sistema,
    int software)
```

Members inherited from class AbstractDAOJDBCTemplateImpl

`com.indra.iquality.dao.jdbctemplateimplem.AbstractDAOJDBCTemplateImpl` (in 6.1, page 292)

- `protected` **dataSource**
- `protected` **helper**
- `public void` **setDataSource**`(javax.sql.DataSource` **dataSource**`)`

6.14 Class TraceOfRegisterDAOJDBCTemplateImpl

Implementation of `TraceOfRegisterDAO` (in 5.10, page 289) using JDBC to connect to an Oracle DB.

Declaration

```
public class TraceOfRegisterDAOJDBCTemplateImpl
 extends com.indra.iquality.dao.jdbctemplateimplem.
    AbstractDAOJDBCTemplateImpl implements com.indra.iquality.dao.
    TraceOfRegisterDAO
```

Field summary

logger The Constant logger.

Constructor summary

TraceOfRegisterDAOJDBCTemplateImpl()

Method summary

getAll(int, String, int)

Fields

- `private static final org.slf4j.Logger` **logger**
  - The Constant logger.

Constructors

- **TraceOfRegisterDAOJDBCTemplateImpl**

  ```
  public TraceOfRegisterDAOJDBCTemplateImpl()
  ```

Methods

- **getAll**

  ```
  java.util.List getAll(int idOperacion,java.lang.String sistema,
      int software) throws java.lang.Exception
  ```

  - **Description copied from com.indra.iquality.dao.TraceOfRegisterDAO** (**in 5.10, page 289**)

    Gets all the register traces of an operation for a given system and software version.

  - **Parameters**
    - `idOperacion` – the identifier of the operation
    - `sistema` – the system
    - `software` – the software version
  - **Returns** – the all

– **Throws**

* `java.lang.Exception` – all the register traces of the operation

Members inherited from class AbstractDAOJDBCTemplateImpl

`com.indra.iquality.dao.jdbctemplateimplem.AbstractDAOJDBCTemplateImpl` (in 6.1, page 292)

- `protected` **dataSource**
- `protected` **helper**
- `public void` **setDataSource**`(javax.sql.DataSource` **dataSource**`)`

6.15 Class ValidacionTecnicaDAOJDBCTemplateImplTest

Declaration

public class ValidacionTecnicaDAOJDBCTemplateImplTest
extends java.lang.Object

Field summary

environment
logger

Constructor summary

ValidacionTecnicaDAOJDBCTemplateImplTest()

Method summary

testGetAll()

Fields

- `private static final org.slf4j.Logger` **logger**

- `private static final com.indra.iquality.singleton.Environment` **environment**

Constructors

- **ValidacionTecnicaDAOJDBCTemplateImplTest**

  public ValidacionTecnicaDAOJDBCTemplateImplTest()

Methods

- **testGetAll**

  public void testGetAll()